12-2016

Designing & Implementing a Java Web Application to Interact with Data Stored in a Distributed File System

Punith Reddy Etikala
St. Cloud State University, petikala@stcloudstate.edu

Designing & Implementing a Java Web Application to Interact with Data Stored in a Distributed File System

by

Punith Reddy Etikala

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

December, 2016

Starred Paper Committee:
Dr. Dennis Guster, Chairperson
Dr. Lynn Collen
Dr. Keith Ewing

Abstract

Every day there is an exponential increase of information and this data must be stored and analyzed. Traditional data warehousing solutions are expensive. Apache Hadoop is a popular open source data store which implements map-reduce concepts to create a distributed database architecture. In this paper, a performance analysis project was devised that compares Apache Hive, which is built on top of Apache Hadoop, with a traditional database such as MySQL. Hive supports HiveQueryLanguage, a SQL like directive language which implements MapReduce jobs. These jobs can then be executed using Hadoop. Hive also has a system catalog – Metastore which is used to index data components. The Hadoop framework is developed to include a duplication detection system which helps managing multiple copies of the same data at the file level. The Java Server Pages and Java Servlet framework were used to build a Java web application to provide a web interface for the clients to access and analyze large data sets present in Apache Hive or MySQL databases.

Acknowledgement

This research paper about designing and implementing Java web applications to interact with data stored in a distributed file system was undertaken using resources provided by the Business Computing Research Laboratory of St. Cloud State University. Data used for the analyses came from the St. Cloud State University library.

Table of Contents

Page

Page

Page

List of Tables

Table Page

List of Figures

Figure Page

Figure Page

Figure Page

Chapter 1: Introduction

Introduction

The interface to any data is critical to being able to use and understand that data. The interface design is particularly important when working in the new area of Big Data. The concept of "Big Data" presents a number of challenges to Information System professionals and especially Web designers. In fact, one of the leading software analytic companies has discretely broken them into five categories as summarized below:

1. Finding and analyzing the data quickly
2. Understanding the data structure and getting it ready for visualization
3. Making sure the data is timely and accurate
4. Displaying meaningful results (for example, using cluster analysis rather than plot the whole data set)
5. Dealing with outliers (how to ensure they get proper attention) (SAS, 2015)

All of the categories are important in obtaining success in using Big Data and Data Analytics. However, this paper will focus primarily on the categories related to finding and extracting data from a potential distributed file system and being able to visualize it in a timely manner.

The work of Jacobs, 2009 puts this into perspective and specifically states that as your data set size grows, the probability that your applications that use that data will become untenable from a performance perspective increases as well. This is particularly true if a Web interface is used to aid in data visualization. Further, Jacobs, 2009 explains that the decay in performance is caused by several factors and each case needs to be carefully assessed and the

application involved tuned to ensure adequate performance. One important example he cites deals with the capabilities of traditional relational databases. Specifically, he states that: it's easier to get the data in than out. Most databases are designed to foster efficient transaction processing like inserting, updating, searching for, and retrieving small amounts of information in a large database.

It appears that platforms have been created to deal with the mass and structure of Big Data. Further, as one might expect they utilize distributed processing as well as software optimization techniques. An excellent summary of this work is presented by Singh & Reddy, 2014. In this work they discuss both horizontal distributed file systems such as Hadoop (and its successor Spark) and vertical systems that rely on high performance solutions which leverage multiple cores. This paper will focus on a specific horizontal system, Hadoop, because the goal herein is to assess performance characteristics of that system when compared to a traditional MYSQL DBMS in cases in which they are accessed via a Web interface. The Hadoop file system and its associated components create a complex, but efficient architecture that can be used to support Big Data analysis. Further, a modular approach can be employed with the Hadoop architecture because a Web interface can interact with Hive (the query module) and efficient performance can be obtained by using the map reduce function that allows Hadoop to function as a distributed file system that can run in parallel.

It is worthwhile to look at the suggested architecture for the Hadoop based data analytics ecosystem and compare it with the traditional scientific computing ecosystem. The work of Reed & Dongarra, 2015 on Exascale computing explains this quite well. This work delineates in detail all the components in the Hadoop architecture, including the map-reduce optimization software.

Because of the interaction with the Web interface the explanation of Hive which is a MapReduce wrapper developed by Facebook, Thusoo et al., 2009, is also useful. This wrapper is a good match for the Web interface design because of its macro nature which makes the coding easier because programmers don't need to directly address the complexities of MapReduce code. Ultimately, if analytics are required via the Web interface the Hadoop based data analytics ecosystem could be considered a unified system because it includes innovative application level components such as R, which is an open source statistical programming language, which is widely used by individual researchers in the life sciences, physical sciences, and social sciences, (Goth, 2015). Goth further states that having a unified system makes the discovery process faster by "closing the loop" between exploration and operation, which reduces the potential for error when compared to a different systems approach. Interestingly, there is a trend to make data scientists responsible for both exploration and production. This paper addresses the production issue by integrating a Web interface.

Big data is a field that is still growing. Some of the areas that are still emerging are improving the storage solutions, access times and optimization software will be explored by data scientists. Najafabadi et al., 2015, felt that relevant future work might involve research related to: defining data sampling criteria, domain adaptation modeling, defining criteria for data sampling and obtaining useful data abstractions, improving semantic indexing, semi-supervised learning, and active learning. Certainly, active learning could benefit from the optimized use of Web interfaces.

In sum, this paper will use a Hadoop based data analytics ecosystem to support the design, implementation and optimization of a Big Data application. Further, to assess its potential

advantages and its performance this system will be compared to a traditional DBMS using the same Web interface. Special attention will be paid to the additional overhead a Web interface places on the system.

This additional overhead is often misunderstood and only evaluated from a single dimension. To understand the full effect overhead one needs to look at the total response time model which is quite complex and involves a number of components.

A good representative example of this model is offered by Fleming, 2004: User -Application-Command _CPULocalComputer _NICLocalComputer _Network-Propagation _Switch _Network-Propagation _Switch _Network-Propagation _NICFile-Server _CPUFile-Server _SCSI-bus _DiskRead then traverse the path in reverse for the reply.

When evaluating this model in terms of a Web based interface to a distributed file system the additional delay caused by the Web service application coupled with the added network load can have a detrimental effect on performance. Evaluating the extent that this occurs is one of the primary goals of this paper.

Problem Statement

Given that the literature review indicates that Big Data is here to stay and the analysis of such data in a timely manner will continue to be problematic there is a need to conduct performance related research. Further, the state of the current technology requires a fair amount of sophistication on the part of an end-user to deal with the parallelization often invoked to provide the desired speed up.

Therefore, this paper will use a Hadoop test-bed with live data to test the performance of a Web interface devised using the Java, JSP framework when deployed using both a Hadoop and a

traditional MySQL database. The primary metric will be elapsed time from client to server which will allow measurement of end-to-end delay and provide a user interface to execute queries on databases and export results of their analysis based on user access level.

Definition of Terms

Table1: Definition of Terms

HDFS	Hadoop Distributed File System
YARN	Yet Another Resource Negotiator
GUID	Global Unique Identifier
DBMS	Database Management System
PK	Primary Key
FK	Foreign Key
Hadoop	Framework that allows for the distributed processing and storage of very large data sets
Hive	Data warehouse
MapReduce	Distribute work around a cluster
DSA	Digital Signature Algorithm
JSP	Java Server Pages

Chapter 2: Literature Review and Background

Challenges of working with Big Data

Because of the large volume of data involved, there are many challenges when working in the area of Big Data. The complexity which is described by Jagadish et al., 2014. Specifically, they state that working in the area of Big Data is a multi-step process and it is important not to ignore any of the steps. In the case of this paper, of course the important step would be to evaluate the interaction of the Web interface with the underlying distributed file system. Jagadish et al., 2014, identified the following required steps: acquisition, information extraction, data cleansing, data integration, modeling/analysis, interpretation and reporting. Too often one or more of the steps are ignored and too much focus is placed on the reporting phase and the "visualization of the results" which often can result in erroneous reporting. Therefore, the Web interface devised and tested herein, will need to be evaluated in terms of accuracy and reliability as well.

Of course, many of the challenges stems from the complex computing environments required to support Big Data. There is a real challenge finding analysts with the technical maturity level needed to support the acquisition and data integration steps that are critical before data modeling can even take place (Morabito, 2015). So therefore, an understanding of the Exascale computing structure described earlier by Reed & Dongarra, 2015 is crucial in being successful in the early steps of data analytics. In the case of this paper, the architecture needs to be expanded to encompass a Web interface.

Need for Distributed File System

The increased volume of data that results from a Big Data concept drastically complicates analytic endeavors. That is not to say that traditional methods of accessing and managing large data sets still have validity and usefulness. With a Web interface there may be a need to support millions of hit scenarios. A primary limitation of traditional methods is that they are often not scalable and may involve additional data set types, particularly unformatted data (Hu, 2014). Traditional data storage methods offer a starting point and can be imported into newer distributed systems so that scalability and adequate access performance can be realized. Generally speaking, the new system would rely on some type of distributed processing and would include concepts such as: ETL (extract, transform and load), EDW (enterprise data warehouse) SMP (symmetric multi-processing) and distributed file systems (such as Hadoop). Obviously, while distributed systems bring more processing power to the table it is critical that there is software in place to manage the multiple threads that will be generated. This is provided in the case of Hadoop by the map-reduce function. Multi-threading is also critical within the Web interface as well, so that if need be a million of hits scenario can be supported.

While not part of the operational research undertaken herein, there are other options to address the extraction logic from distributed data stores. A very popular option is the concept of NoSQL databases. Traditional relational model database imposes a strict schema, this is in contrast to many of the concepts within Big Data which are based on data evolution and necessitate scaling across clusters. Thus, NoSQL databases support schema-less records which allow data models to evolve (Gorton & Klein, 2014). The four most prominent data models within this context according to Gorton & Klein, 2014 are:

1. Document databases (often XML or JSON based in MongoDB)
2. Key-value databases (such as Riak and DynamoDB)
3. Column-oriented databases (such as HBase and Cassandra)
4. Graph databases (such as Neo4j and GraphBase)

Architectures to support Big Data

There is much support for the concept of a distributed file system offering an effective platform to support Big Data. While there may be other viable options in terms of design or functionality, but distributed file systems by far offer the most cost effective solution (Jarr, 2014). A prime example of this is Hadoop, which is designed to deploy a distributed file system on cheap commodity machines (Reed & Dongarra, 2015).

It also is interesting to note that the architecture to capture the Big Data in the first place is expanding as well. This environment is personified by the Internet of Things (IoT) concept. It relies on interconnected physical objects which effectively creates a mesh of sensor devices capable of producing a mass of stored information. These sensors based networks pervade our environment (e.g., cars, buildings, and smartphones) and continuously collect data about our lives (Cecchinel, Jimenez, Mosser & Riveill, 2014). Thus, the use of it will further propagate the legacy of Big Data.

Performance Issues with Big Data

Performance issues in Big Data stem from more than just the large amounts of data involved. However, Big Data is characterized by other dimensions as well. Jewell et al., 2014 has actually identified four dimensions:

1. Volume (Big data applications must manage and process large amounts of data),

2. Velocity (Big data applications must process data that are arriving more rapidly),
3. Variety (Big data applications must process many kinds of data, both structured and unstructured) and
4. Veracity (Big data applications must include a mechanism to assess the correctness of the large amount data of rapidly).

These dimensions provide multiple parameters from which to tune a system from a performance perspective. Therefore, the computing environment required will need to be adept in dealing with real-time processing, complex data relationships, complex analytics, efficient search capabilities as well as effective Web interfaces. Given the current options, a private cloud could be configured to maximize both processing speed as well as IO movement. Such a cloud would lean heavily on distributed processing, distributed file systems and multiple instance of the Web service. Further, dynamic allocation of resources would need to be implemented as well. This might involve multiple instance of the Web service replicated across multiple hosts with load balancing invoked.

The work of Jacobs, 2009 is very useful in putting the concept of Big Data into perspective. He states that people often expect to extract data in seconds that took months and months to store. So one could interpret this to mean it is a lot easier to get data into a traditional relational database then get it out. It can be treated as a mass storage device and “chunks” of the total can be extracted for partial processing, but when the Big Data is analyzed in bulk, the scalability is not there and performance takes a nose dive. This is further compounded when Web interfaces are involved. He further states that anticipating what “chunks” are needed and extracting them to a data warehouse can help, but optimizing systems to use their full processing

and IO capabilities is challenging. For example, with reasonable numbers of transactions random processing can be advantageous. However, when using mechanical drives an analysis algorithm that utilizes random access memory may actually run slower than the same data can be read in sequence from a mechanical drive. Without a doubt, the concept of distributed file systems is a step in the right direction, but they too have limitations such as network latency. This network latency may further complicate the performance of the Web interface if the client connection is sharing the same network with a distributed file system. Hence sound network design within a private cloud is critical. It will then be necessary for future systems to encompass designs that expand the boundaries of current day thinking. No doubt the analyses of huge datasets will become routine. A ramification of this case is that analysts that will be successful in analyzing those data sets will need to look beyond off-the-shelf techniques and implement techniques that take advantage of the environmental architecture (such as cloud computing), optimize the hardware resources and devise/implement algorithms designed specifically to deal with Big Data in an optimized hardware environment. Of course, if a Web interface is the entry point of that system, it will need to be optimized and properly secured too.

It has been established that the volume of processing within Big Data requires a well-designed architecture if reasonable performance is to be obtained. As previously stated the work of Reed & Dongarra, 2015, provides excellent insight into Exascale computing. A foundation for this architecture is the concept of a distributed storage system which would allow the data to be extracted from multiple devices simultaneously (Chang et al., 2008). Of course the Hadoop file system follows this logic. One of the benefits of Hadoop is that it is basically a data-analytics cluster that can be based on commodity Ethernet networking technology and numerous PC nodes

(even a generation or two old) containing local storage. This model goes a long way in providing a cost effective solution for large scale data-analytics (Lucas et al., 2014). Hadoop could then be viewed as the logic fabric to bind them together. This characteristic made it easy to create a test-bed environment for this paper. In fact, the resources needed were quickly configured in the author's private cloud using virtualization software.

A key component in the Hadoop system implementation is the Map Reduce model (Dean & Ghemawat, 2004). First of all, Map Reduce is designed to facilitate the parallel processing function within Hadoop applications. It is designed to utilize multi-core as well as processors distributed across multiple computing nodes. The foundation of the Map Reduce system is a distributed file system. Its primary function is based on a simple concept: Large files are broken into equal size blocks, which are then distributed across, in our case, a commodity cluster and stored. In our case the storage was within a private cloud and it was critical to implement fault tolerance so therefore each block was stored several times (at least three times) on different computers nodes.

A challenge with undertaking a performance analysis of this type is dealing with new technology and learning new things. The authors' primary background in deal with large data sources was a traditional relational data base structure. Fortunately, a couple of tools are available to assist in extracting data from the Hadoop file system. First there is "PIG" which was devised by Yahoo! to streamline the process of analyzing large data sets by reducing the time required to write mapper and reducer programs. According to IBM 2015b, the pig analogy stems from actual pigs, who eat almost anything, hence, the PIG programming language is designed to handle any kind of data! While it boasts a powerful programming language it is basically new syntax and requires

time to master. Another option Hive, uses an SQL derivative called Hive Query Language (HQL) so that the developer is not starting from scratch and has a much shorter learning curve. While HQL does not have the full capabilities of SQL it is still pretty useful (IBM, 2015a). It completes its primary purpose quite well which is to serve as a front end to simplify MAP REDUCE jobs that are executed across a Hadoop Cluster.

Advantages of Hadoop and Map Reduce

While the cloud architecture makes available numerous dynamically allocated resources there must be some type of strategy to be able to multi-thread applications in a cost effective manner. Hadoop is able to provide that efficient and cost effective platform for distributed data stores. A key component is the Map Reduce function (MR). This function provides the means to connect the distributed data segments in a meaningful way and take advantage of parallel processing to ensure optimum performance. Clearly, the primary goal is to use these components to facilitate the analysis of Big Data in a timely fashion. For applications that might still run in a relational world, MR can also be used with parallel DBMS. In cornerstone applications like ETL systems it can be complementary to DBMSs, since databases are not designed to be efficient at ETL tasks.

In a benchmark study using a popular open-source MR implementation and two parallel DBMSs, Stonebraker et al., 2010 found that DBMSs are substantially faster than MR open source systems once the data is loaded, but that loading the data takes considerably longer to load in the database systems. Dean & Ghemawat, 2010 expanded on the interrelationship between MapReduce and parallel databases and found MR provides many significant advantages over parallel databases. First and most important, MR provides fine-grain fault tolerance for large jobs and accordingly

failure in the middle of a multi-hour execution does not require restarting the job from scratch. This would be especially important for Web interfaces that typically do not have check pointing built in. Second, MR is most useful for handling, data processing and data loading in a heterogeneous system with numerous varied storage systems (which describes a private cloud). Third, MR is an excellent framework to support the execution of more complex functions than are not directly supported in SQL. In summary, MR can be an effective means of linking complex data parts together no matter the architecture, but is especially effective when used in conjunction with Hadoop (Reed & Dongarra, 2015).

Challenges of Effectively Integrating a Web with Hadoop and Map Reduce

As one would expect the distributed nature of Hadoop complicates devising an effective Web interface. The motivation for the Web interface is to allow less technical people to be able to get around submitting static pieces of code from the command line. Of course that code will have to deal with components such as the mapper and a reducer.

While there is a rudimentary Web interface that allows the submission HiveQL statements it lacks the depth needed for sophisticated analysis. This is included with Apache as part of the Hive source tarball.

Devising a sophisticated Hadoop Web Interface requires a different approach. According to Logical Thoughts on Technology, 2013 the first step is to create a generic job submitter, one that can then be used in a service call in the Web application. This user interface (UI) would present some nice, clean, easy to use interface, next the user would make same sequence of selections, and then they would click a button to start their job. It therefore follows then that on the back-end, the request would be passed to a service call where the parameter set would be

processed and turned into a Hadoop job, and thereby submitted to the cluster. Logical Thoughts on Technology, 2013 summarizes the three processing components as follows:

1. Something to gather up the set of parameters for each job
2. Something to convert string class names into actual classes
3. Something to step through the parameters, then perform any formatting/processing, and submit the job

Last, they suggest that the production Web application that will perform the suggested function be written in Java so that class objects are easily obtainable in accordance with sound OOPs programming.

While the advantages of using Java for production Web applications are well known, it is appropriate to provide a brief explanation of the "Java Server Pages framework" which will be used to devise the Web interface for this project. As stated earlier, the cloud computing environment can be quite complex. One of the primary purposes of the JSP framework is to create a transparent interface to the infrastructure so that the programmer can more easily focus on the application.

This brings us to the last topic of the Literature Review which deals with providing adequate performance. A typical industry based standard for acceptable performance is a client response time of three seconds or less. This is challenging under the best of circumstances, but even more elusive in the world of Big Data.

As state earlier in the work of Fleming, 2004 indicated that it was just not how quickly data could be read from the data source, but rather the result of end-to-end delay that an end-user is concerned about. Guster, O'Brien & Lebentritt, 2013 address this as follows: "Given that the

network delay on the Internet in the US might often take .5 seconds in each direction it is important to optimize each of the parameters. Further, one needs to realize that this whole algorithm is based on queuing theory, which means that there is an interaction among all the parameters. In other words, a delay of .0005 instead of .0001 at the first parameter won't simply result in .0004 seconds of additional response time. Rather, it will propagate through the entire algorithm and the delay will get a little longer with added wait time at each successive parameter. To put this in perspective, if one assumes a geometric progression through all 12 parameters in the algorithm above (Fleming, 2004) the result in total added delay would be close to 1 second (.8192)".

While Guster, O'Brien & Lebentritt, 2013 did realize acceptable performance in regard to providing a Web interface delay of three seconds or less they were working in a less stringent environment. They were using Casandra rather than Hadoop (which integrates the MR) function, the volume of data was much less and they were not working in a true cloud computing environment. It will be interesting to compare performance metrics between the systems and with a traditional database.

Chapter 3: Methodology

Design of the Study

The following diagram explains the Hadoop architecture used for this project. It is centered on the HDFS file system which is used to store large data files. To achieve the desired parallelism MapReduce and the YARN framework is used to process HDFS data and provide resource management. Apache Hive is built on top of Hadoop to provide a data summarization and analysis on HDFS data. Apache Sqoop is used to transfer data between relational databases and the HDFS file system. The Java Web application is originated by using JSP and Servlet framework and allows reports to be displayed and provides the Web application needed to compare performance times between MySQL and Hive. Similar architecture referred by Afreen, 2016 to work on design and performance characteristics.

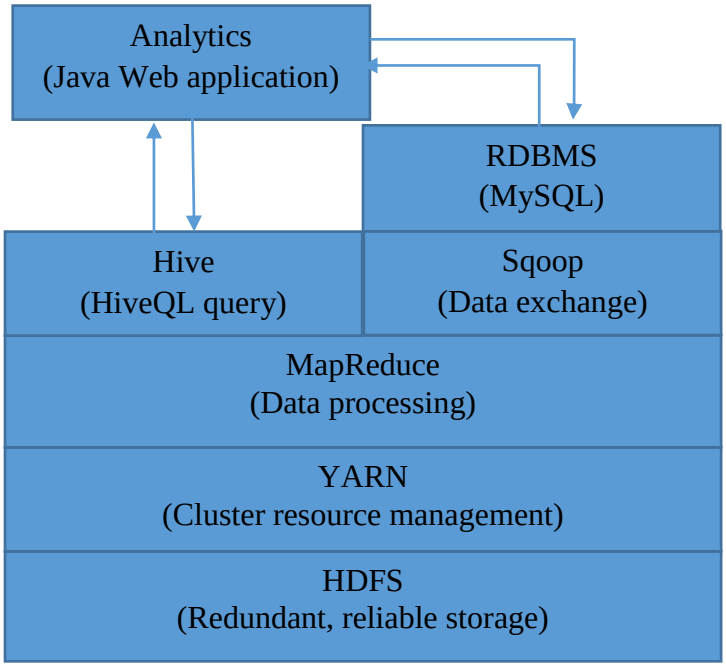


Figure 1: Hadoop Architecture

Web Application Technical Architecture

The user interface is created by using Java Server Pages. When a user submits or calls some function like executing queries on a MySQL or Hive database a comparison of execution time between the MySQL and Hive queries is recorded. These function calls might include: Displaying a table structure, Generating charts etc. The interface then calls Action Classes, Servlets generally, which call Service functions and then the Data Access Object layer to connect to the database to pass the results to Java Server Pages. Generally, the results are displayed in table structure as well as Line charts and provide a basic synopsis of performance regarding query execution on MySQL and Hive databases.

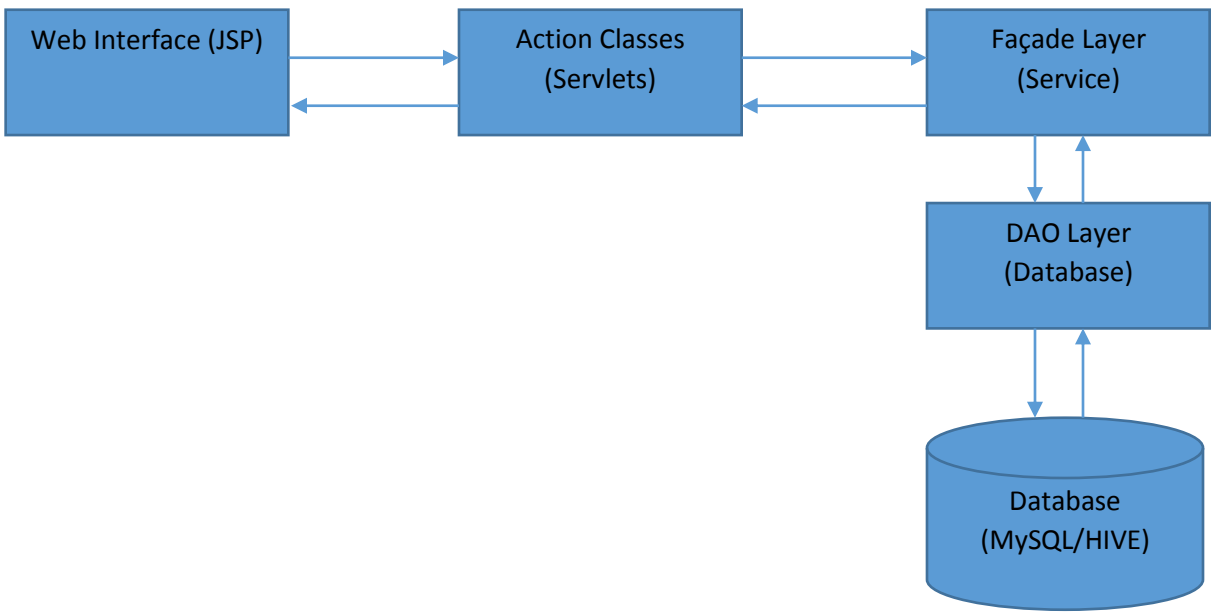


Figure 2: Web Interface Technical Architecture

Data Collection

The data has been collected from two resources

1. Saint Cloud State University campus library

The original data used in this project were provided by Saint Cloud State University campus library. This data is available in a MySQL database. It contains three tables: Students, Majors and CirculationLog. The Students table contains the student's basic information and each student can be distinguished by a unique GUID key, UniqueId and there is also a UniqueStudentId which acts as the primary key. The Majors table has information about students registered for a particular major. The Majors table is linked to the Students table by a foreign key, UniqueStudentId and UniqueMajorId act as the primary key for this table. The CirculationLog table is linked to the Students table by the foreign key, UniqueStudentId and UniqueCirculationId act as the primary key for this table. For Hadoop analysis, the data in the CirculationLog table has been regenerated many times to increase its volume, which validates the big data concept within Hadoop. The following table shows in detail information about the Students, Majors and CirculationLog tables.

Table 2: Students table in MySQL

Field	**Type**
UniqueStudentId (Primary Key)	INT
UniqueId	VARCHAR
QPP	FLOAT
HS_GPA	DECIMAL
HS_GPAScale	DECIMAL
HS_Rank	INT
HS_GraduationDate	DATETIME
HS_Name	VARCHAR
HS_Code	VARCHAR
HS_City	VARCHAR
HS_State	VARCHAR
HS_Zip	VARCHAR
HS_MnSCURegion	VARCHAR
HS_District	VARCHAR
HS_DistrictCode	VARCHAR
ACTScore	DECIMAL
LibraryUsed	BOOLEAN

Table 3: Majors table in MySQL

Field	**Type**
UniqueMajorId (Primary Key)	INT
UniqueId	VARCHAR
Major	VARCHAR
MajorCode	VARCHAR
MajorProgram	VARCHAR
MajorDepartment	VARCHAR
MajorSchool	VARCHAR
MajorCollege	VARCHAR
FY	VARCHAR
UniqueStudentId (Foreign Key)	INT

Table 4: CirculationLog table in MySQL

Field	**Type**
UniqueCirculationId (Primary Key)	INT
UniqueId	VARCHAR
YearTerm	VARCHAR
TermName	VARCHAR
Date	VARCHAR
DateOfTerm	INT
Hour	VARCHAR
Action	VARCHAR
Id	VARCHAR
Budget	VARCHAR
Profile-id	VARCHAR
Barcode	VARCHAR
Material	VARCHAR
Item-status	VARCHAR
Collection	VARCHAR
Description	VARCHAR
Doc-title	VARCHAR
UniqueStudentId (Foreign Key)	INT

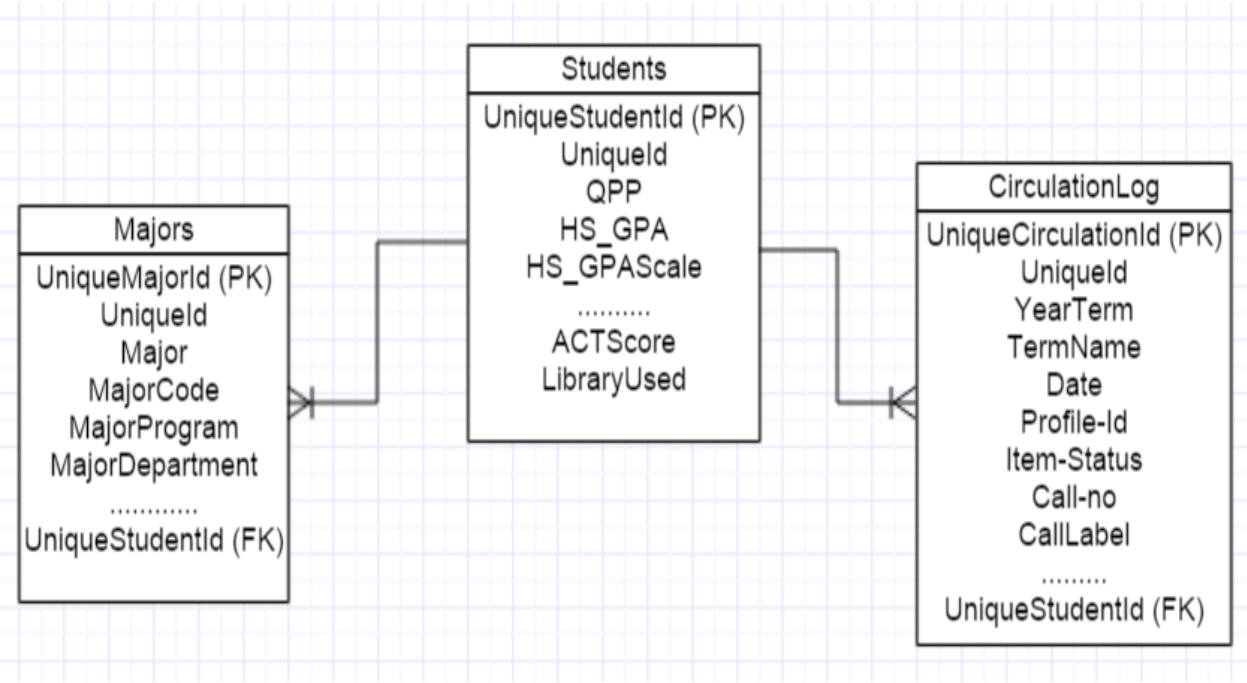


Figure 3: Data Model in MySQL

2. Twitter App

This application allows additional Big Data to be downloaded in real-time data from the Twitter company's server. "https://apps.twitter.com/" internet site allows for us to create a Twitter App. Refer Twitter API Overview for details.

In the Application Management window, "Create New App" allows us to make an application.

Figure 4: Twitter Application Management

Once the application is created successfully, On the Application Management screen, the newly created Twitter Application appears.

Figure 5: Twitter Application

To spread out the newly created Twitter Application one must sail to the "Keys and Access Tokens" tab, where Consumer Key also called as Application Programming Interface Key, Consumer Secret or Application Programming Interface Secret Key, Access Token Key and Access Token Secret Key are the 4 secret keys. These keys allow the Java program to connect to the Twitter Application to retrieve the data from the Twitter company server.

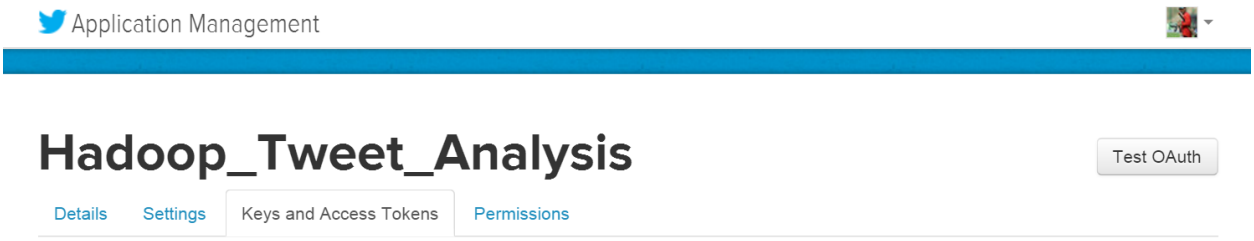


Figure 6: Twitter Application Key and Access Tokens Management

Refer to Appendix A for the Java source code, which fetches data and writes it in a local file. To perform analysis with Hadoop, 160GB of data was gathered from the Twitter server. TwitterFeeds.java, available in Appendix A is used to download raw data, which is in json format which is consistent with the object oriented programming approach used herein. Converter.java, available in Appendix A is used to parse the content within the json object and

gather the required data to perform analysis with Hadoop. There are four main “objects” within the API: Tweets, Users, and Entities (see also Entities in Objects), and Places in the feeds.

Tools and Techniques

Apache Sqoop

Sqoop is a tool developed to transfer data between Hadoop databases and relational databases. “Sqoop is used to import data from a relational database management system (RDBMS) such as MySQL or Oracle or a mainframe into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS”. Here, Sqoop is used to transfer data from a MySQL database to HDFS. Refer Sqoop User Guide (v1.4.6) for more details.

MySQL

The MySQL software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. It is a special purpose programming language which was designed for managing data held in relational databases. This has a wide scope of functions including: data insert, delete, query, update, and schema creation and modification functions. Here, MySQL acts as traditional database with large data. Refer A Quick Guide to Using the MySQL APT Repository for more details.

Apache Tomcat

This acts a web server to host the project. Apache Tomcat Servlet/JSP container acts as an entry point of the documentation bundle. Apache Tomcat is a platform for developing and deploying web applications and web services. Tomcat is an open source web server developed

by Apache Tomcat Foundation released under Apache License. Here, Tomcat acts as a web application server to support the project. Refer Apache Tomcat 8 for more details.

Java Server Pages Framework

The JSP framework is built on top of a Java Servlet API, it provides tag based templates, follows the server programming model and it is document centric. The Java code can be compiled and executed when a request is received in JSP. Here the JSP framework is used to develop enterprise web applications which allow end users to connect to the Hadoop Hive database/MySQL database and analyze their performance when dealing with large data sets.

Apache Hive

The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. Here Hive acts as data warehouse which stores HDFS data and HiveQL allows users to perform queries against the HDFS, which starts MapReduce jobs to fetch the results by implementing a multi-threading approach. Refer Apache Hive for more details.

Hadoop Key Terms

Namenode: is the centerpiece of the Hadoop file system. Namenode records the structure of all files in the file system and keeps tracks of where the file data is stored.

Datanode: is the data store in HDFS. A cluster can have more than just a single data node and can implement more than one data replication across them.

SecondaryNamenode: is a dedicated node in Hadoop, whose function is to take checkpoints of file system metadata information present in Namenode.

JobTracker: is a service within Hadoop that runs Hadoop MapReduce jobs on specific nodes in the cluster.

TaskTracker: JobTracker creates tasks like Map, Reduce and Shuffle operations for the TaskTracker to perform.

ResourceManager: Manages the distributed applications running and keeps master lists of all resource attributes across the Hadoop Cluster.

NodeManager: It is responsible for individual computer nodes in a Hadoop Cluster. It is YARN's management node agent.

Shell Commands

sudo: sudo allows the users to execute the commands in superuser or other users, whose accounts are present in sudoers file. By default, sudo asks for a password to authenticate and allows the user to execute the commands in superuser or another user form for a specific time.

apt-get: apt stands for Advanced Packing tool. "apt-get" is used to install the new software packages, upgrade the existing software packages and update the package list index. It has many advantages over other Linux management tools that are available.

which: the which command in the Linux atmosphere is used to show the full path of the commands.

wget: wget is the command which allows one to download the files or software without interaction from the user which means it doesn't need the user to be logged in and accordingly the wget command runs in the background. It supports HTTP, HTTPS and FTP protocols.

tar: the tar command is used for archiving files, which means storing or extracting the files from an archive file which has a .tar extension.

mv: the mv command is used to move a file from its source to a directory or to rename a file from source to destination.

cp: the cp command is used to copy a file from its source to a directory.

nano: the nano command is used to simply open and edit the contents of the file or create a new file and save the file, even though “vi” and “emacs” does the same work, nano is a simple command which can be run without any options.

source: the source command is used to evaluate a file or resource such as a tcl script. It takes the given contents and passes it to the tcl interpreter which returns the command if it exists. If an error occurs, it simply returns the error.

ln: the ln command is used to make links between files.

chown: chown is used to change the ownership of a file. Only the super user can change the ownership of any file. fchown, lchown also fit into the same category.

Hardware Environment

Four Virtual Machines using the Ubuntu 14.04.3 Operating System

Table 5: Virtual Machine Details

IP Address	**Number of Cores**	**RAM**	**CPU Clock Speed**	**Node Name**
10.59.7.90	8	16GB	2200Mz	masternode
10.59.7.91	2	4GB	2200Mz	datanode1
10.59.7.92	2	4GB	2200Mz	datanode2
10.59.7.93	2	4GB	2200Mz	datanode3

Software Environment

1. Java 1.7.0_79
2. OpenSSH 6.6.1
3. MySQL Server 5.6
4. Apache Hadoop 2.6.2
5. Apache Hive 1.2.1
6. Apache Sqoop 1.4.6 – For Hadoop 2.x
7. Apache Tomcat7
8. Eclipse IDE for Java EE Developers

Chapter 4: Implementation

Installing Java

The following commands will update the package index and install the Java Runtime Environment. Refer The Java EE 5 Tutorial for more details.

```
sudo apt-get update
sudo apt-get install openjdk-7-jre
```

The openjdk-7-jre package contains just the Java Runtime Environment. If one wants to develop java programs, then the openjdk-7-jdk package would need to be installed.

The following command is used to verify that java installed.

```
java -version
```

Installing SSH

There are two components of SSH:

SSH: This command is used to connect to remote client machines, generally invoked by the client.

SSHD: The daemon which runs on the server, allowing the clients to connect to the server.

SSH can be installed by using the following command.

```
sudo apt-get install ssh
```

To locate the pathname for the SSH or SSHD commands the which command may be used.

```
which ssh
/usr/bin/ssh
```

```
which sshd
/usr/sbin/sshd
```

Installing and Configuring Apache Tomcat

One begins by downloading the tomcat binary from the tomcat source repository by using the following command.

```
wget http://mirror.cc.columbia.edu/pub/software/apache/tomcat/tomcat-
8/v8.0.32/bin/apache-tomcat-8.0.32.tar.gz
```

Extract the .tar.gz file and move it to the appropriate location.

```
tar xvzf apache-tomcat-8.0.32.tar.gz
mv apache-tomcat-8.0.32 /opt/tomcat
```

Adding a tomcat home directory to path.

```
sudo nano ~/.bashrc
export CATALINA_HOME=/opt/tomcat
sudo source ~/.bashrc
```

Configuring tomcat user roles by editing tomcat-users.xml.

```
nano $CATALINA_HOME/conf/tomcat-users.xml
<tomcat-users>
        <role rolename=”manager-gui”/>
        <role rolename=”admin-gui”/>
        <user username=”<username>” password=”<password>”
        roles=”manager-gui, admin-gui”/>
</tomcat-users>
```

<tomcat-users>: Users and roles are configured.

<role>: Specifies list of roles.

<user>: User's username, password and roles are assigned under this tag. Users can have multiple roles defined through a comma delimited list.

Here, the manager-gui role allows the user to access the manager web application (http://localhost:8080/manager/html) and the admin-gui role allows the user to access the host-manager web application (http://localhost:8080/host-manager/html).

By default, the tomcat server runs on port 8080, but this can be changed by modifying the server.xml file in the $CATALINA_HOME/conf folder.

To start the tomcat server:

$CATALINA_HOME/bin/startup.sh

To stop the tomcat server:

$CATALINA_HOME/bin/shutdown.sh

Installing and Configuring MySQL Server 5.6

MySQL installation is made simple by using an ‘apt-get’ command. Open the terminal window in masternode, and use the following command:

sudo apt-get install mysql-server-5.6

This install the package for the MySQL server, as well as the packages for the client and for the database common files. During the installation, supply a password for the root user for your MySQL installation.

Now, Configure MySQL by editing ‘/etc/mysql/my.cnf’. Bind the masternode ip-address to MySQL server and assign an open port for MySQL to run.

bind-address = <masternode ip-address>

port = 3306

The MySQL server is started automatically after installation.

Check the status of the MySQL server with the following command:

service mysql status

Stop the MySQL server with the following command:

service mysql stop

Start the MySQL server with the following command:

service mysql start

Disabling IPv6

Because Hadoop is not supported on IPv6 networks and has been developed and tested on IPv4 networks, Hadoop needs to be set to only accept IPv4 clients.

Add the following configuration to /etc/sysctl.conf to disable IPv6 networks and restart the current network.

net.ipv6.conf.all.disable_ipv6 = 1

net.ipv6.conf.default.disable_ipv6 = 1

net.ipv6.conf.lo.disable_ipv6 = 1

Installing and Configuring Apache Sqoop

Download the Sqoop binary from the Sqoop source repository by using the following command.

wget http://download.nextag.com/apache/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz

Extract the .tar.gz file and move it to the appropriate location.

```
tar xvzf sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
mv sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz /home/student/sqoop
```

Changing the owner and group for the Sqoop installation directory to the Hadoop dedicated user.

```
sudo chown -R student:hadoop /home/student/sqoop
```

Adding the sqoop home directory and sqoop binary directory to path.

```
sudo nano ~/.bashrc
#SQOOP VARIABLES START
export SQOOP_HOME=<sqoop-home-directory>
export PATH=$PATH:$SQOOP_HOME/bin
#SQOOP VARIABLES END
sudo source ~/.bashrc
```

Configuring the Sqoop environmental variables by using the sqoop-env-template.sh template.

```
mv $SQOOP_HOME/conf/sqoop-env-template.sh $SQOOP_HOME/conf/sqoop-
env.sh
nano $SQOOP_HOME/conf/sqoop-env.sh
export HADOOP_COMMON_HOME=<hadoop-home-directory>
export HADOOP_MAPRED_HOME=<hadoop-home-directory>
```

Adding the mysql-connector-java.jar to Sqoop libraries.

```
sudo apt-get install libmysql-java
```

ln -s /usr/share/java/mysql-connector-java.jar $SQOOP_HOME/lib/mysql-connector-java.jar

The following command is used to verify the Sqoop version.

sqoop-version

Installing and Configuring Apache Hive

Download the Hive binary from the Hive source repository by using the following command.

wget http://ftp.wayne.edu/apache/hive/stable/apache-hive-1.2.1-bin.tar.gz

Extract the .tar.gz file and move it to the appropriate location.

tar xvzf apache-hive-1.2.1-bin.tar.gz

mv apache-hive-1.2.1-bin.tar.gz /home/student/hive

Changing the owner and group for the Hive installation directory to the Hadoop dedicated user.

sudo chown -R student:hadoop /home/student/hive

Adding the hive home directory and the hive binary directory to the path.

sudo nano ~/.bashrc

#Hive VARIABLES START

export HIVE_HOME=<hive-home-directory>

export PATH=$PATH:$HIVE_HOME/bin

#Hive VARIABLES END

sudo source ~/.bashrc

Configuring the hive environmental variables by adding HADOOP_HOME to hive-config.sh.

```
nano $HIVE_HOME/bin/hive-config.sh
export HADOOP_HOME=<hadoop-home-directory>
```

Adding the mysql-connector-java.jar to the Hive libraries.

```
sudo apt-get install libmysql-java
ln -s /usr/share/java/mysql-connector-java.jar $HIVE_HOME/lib/mysql-connector-java.jar
```

Now one can configure the Hive metastore service, where the Metastore service provides the interface to Hive and the Metastore database stores the mappings to the data and data definitions. It is important to edit the hive-site.xml file within the conf directory of Hive, so that the MySQL database acts as the Metastore database for Hive.

```
cp $Hive_HOME/conf/hive-default.xml.template $Hive_HOME/conf/hive-site.xml
nano $Hive_HOME/conf/hive-site.xml
<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://10.59.7.51/metastore_db?createDatabase
    IfNotExist=true</value>
    <description>Metadata is stored in a MySQL server</description>
</property>
<property>
```

```
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.jdbc.Driver</value>
        <description>MySQL JDBC driver class</description>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>hiveuser</value>
        <description>Username to connect to MySQL server</description>
    </property>
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>hivepassword</value>
        <description>Password to connect to MySQL server</description>
    </property>
```

Configure Metastore in MySQL by creating the metastore_db database and upgrade the tables by using the hive schema for the MySQL database.

```
mysql -u <username> -p
Enter password: <password>
mysql> create database metastore_db;
mysql> use metastore_db;
mysql> SOURCE $Hive_HOME/scripts/metastore/upgrade/mysql/hive-schema-
0.14.0.mysql.sql;
```

mysql> CREATE USER 'hiveuser'@'%' IDENTIFIED BY 'hivepassword';

mysql> GRANT all on *.* to 'hiveuser'@<machine-ip> identified by 'hivepassword';

mysql> flush privileges;

mysql> exit;

The following command set is used to verify the Hive installation

hive

Installing and Configuring Apache Hadoop

Download the Hadoop binary from the Hadoop source repository by using the following command in all the virtual machines. Refer HadoopIPv6 for more details.

wget http://www-us.apache.org/dist/hadoop/common/hadoop-2.6.1/hadoop-2.6.1.tar.gz

Extract the .tar.gz file.

tar xvzf hadoop-2.6.1.tar.gz

Changing owner and group for the Hadoop installation directory to the Hadoop dedicated user.

sudo chown -R student:hadoop /home/student/hadoop

When configuring Hadoop be aware that it involves numerous files.

1. ~/.bashrc

#Hadoop variables start

export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64

export HADOOP_INSTALL=<Hadoop home directory>

```
export HADOOP_HOME=$HADOOP_INSTALL
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="$HADOOP_OPTS -
Djava.library.path=$HADOOP_HOME/lib/native"
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
#Hadoop variables end
```

2. hdfs-site.xml

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/home/student/hadoop-2.6.1/hadoop_store/hdfs/namenode</value>
    </property>
```

```
 <property>
        <name>dfs.namenode.http-address</name>
        <value>masternode:51070</value>
    </property>
</configuration>
```

3. yarn-site.xml

```
<configuration>
<property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value> org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.resourcemanager.resource-tracker.address</name>
        <value>masternode:8026</value>
    </property>
    <property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>masternode:8031</value>
```

```
    </property>
    <property>
        <name>yarn.resourcemanager.address</name>
        <value>masternode:8051</value>
    </property>
</configuration>
```

4. mapred-site.xml

```
<configuration>
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
        <property>
        <name>mapred.local.dir</name>
        <value>file:/home/student/hadoop-2.6.1/hadoop_store/mapred/local</value>
        <description>Determines where temporary MapReduce data is written. It also may be
a list of directories.</description>
    </property>
    <property>
        <name>mapred.map.tasks</name>
        <value>30</value>
```

<description>As a rule of thumb, use 10x the number of slaves (i.e., number of tasktrackers).</description>

</property>

<property>

<name>mapred.reduce.tasks</name>

<value>6</value>

<description>As a rule of thumb, use 2x the number of slave processors (i.e., number of tasktrackers).</description>

</property>

</configuration>

5. core-site.xml

<configuration>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/student/hadoop-2.6.1/tmp</value>

<description>A base for other temporary directories.</description>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://masternode:54310</value>

<description>The name of the default file system. A URI whose

scheme and authority determine the FileSystem implementation. The

uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.</description>

</property>

</configuration>

6. masters

   student@masternode

7. slaves

   student@datanode1

   student@datanode2

   student@datanode3

Format the Hadoop namenode with the following command

   hadoop namenode –format

To start Hadoop daemons

   start-all.sh

To stop Hadoop daemons

   stop-all.sh

Enabling Secure Connection using SSH

Now keys can be created for the encryption process using the Digital Signature Algorithm and Installing the authorized public key in all the nodes, so that machines can be connected by using SSH.

SSH keys provide a secure way to login to a virtual server. ssh-keygen provides a key pair which generally consists of a public key and a private key. The Public Key is placed on a server and the Private Key is placed on client, which allows server/client communications in a secure way. The following command is used to generate keygen, and then copied to different clients to establish secure communication. The following code is an example for creating keygen in masternode and then copying it to different datanodes to ensure security between/among them.

```
student@masternode:~$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
student@masternode:~$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
student@masternode:~$ ssh-copy-id -i ~/.ssh/id_dsa.pub student@datanode1
student@masternode:~$ ssh-copy-id -i ~/.ssh/id_dsa.pub student@datanode2
student@masternode:~$ ssh-copy-id -i ~/.ssh/id_dsa.pub student@datanode3
```

Configuring Hostname

Hostnames are user readable nicknames that correspond to the IP address of a device connected to the network.

Next configuration of the hostname and mapping the IP addresses to respective hostname in the masternode machine can take place. Similarly, we need to configure the hostname and map IP addresses to the hostnames in all the 3 datanodes.

```
sudo nano /etc/hostname
masternode
sudo nano /etc/hosts
10.59.7.90      masternode localhost
10.59.7.91      datanode1
```

10.59.7.92 datanode2

10.59.7.93 datanode3

Loading Twitter Data from Text file to MySQL

The following command is used to load data from a file structure to a MySQL database by using MySQL connection information and the database name as parameters to the command.

student@masternode:~$ mysqlimport --user=root --password=root --fields-terminated-by='|' --lines-terminated-by='\n' --local hadoopanalysis TweetAnalysis

Importing Data from MySQL to HDFS

The following commands are used to import data from the MySQL database to HDFS file system using Apache Sqoop.

student@masternode:~$ sqoop import --connect jdbc:mysql://10.59.7.90:3306/hadoopanalysis --table TwitterAnalysis --username hiveuser --password hivepassword

student@masternode:~$ sqoop import --connect jdbc:mysql://10.59.7.90:3306/hadoopanalysis --table CirculationLog --username hiveuser --password hivepassword

student@masternode:~$ sqoop import --connect jdbc:mysql://10.59.7.90:3306/hadoopanalysis --table Majors --username hiveuser --password hivepassword

student@masternode:~$ sqoop import --connect jdbc:mysql://10.59.7.90:3306/hadoopanalysis --table Students --username hiveuser --password hivepassword

Creating Tables in Hive

The following commands are used to create tables in Hive.

```
hive> create table TwitterAnalysis(UniqueID BIGINT,TweetID BIGINT, Time_stamp
VARCHAR(255), Tweet VARCHAR(255),FavouriteCount BIGINT, ReTweetCount
BIGINT, lang VARCHAR(255), UserID BIGINT, UserName VARCHAR(255),
ScreenName VARCHAR(255),Location VARCHAR(255), FollowersCount BIGINT,
FriendsCount BIGINT, Statuses BIGINT, Timezone VARCHAR(255))COMMENT
'TwitterAnalysis' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES
TERMINATED BY '\n';

hive> create table CirculationLog (UniqueCirculationId BIGINT, UniqueId varchar(40),
YearTerm varchar(5), TermName varchar(40), CirculationDate varchar(50), DayOfTerm
INT, hour varchar(50), action varchar(50), id varchar(50), budget varchar(50), profile_id
varchar(50), barcode varchar(50), material varchar(50), item_status varchar(50),
collection varchar(100), call_no varchar(50), description varchar(100), doc_title
varchar(500), UniqueStudentId BIGINT, CallLabel varchar(40))COMMENT
'CirculationLog' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES
TERMINATED BY '\n';

hive> create table Majors (UniqueMajorId BIGINT, UniqueId VARCHAR(40), Major
VARCHAR(19), MajorCode VARCHAR(4), MajorProgram VARCHAR(100),
MajorDepartment VARCHAR(100), MajorSchool VARCHAR(100), MajorCollege
```

VARCHAR(100), FY VARCHAR(4), UniqueStudentId BIGINT)COMMENT 'Majors' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';

hive> create table Students (UniqueStudentId BIGINT, UniqueId VARCHAR(40),QPP FLOAT,HS_GPA DOUBLE, HS_GPAScale DOUBLE, HS_Rank INT,HS_GraduationDate TIMESTAMP,HS_Name VARCHAR(100), HS_Code VARCHAR(8), HS_City VARCHAR(40), HS_State VARCHAR(2),HS_Zip VARCHAR(5), HS_MnSCURegion VARCHAR(2), HS_District VARCHAR(50), HS_DistrictCode VARCHAR(8), ACTScore DOUBLE, LibraryUsed BOOLEAN) COMMENT 'Students' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';

Table 6: Students table in Hive

Field	**Type**
UniqueStudentId	BIGINT
UniqueId	VARCHAR
QPP	FLOAT
HS_GPA	DOUBLE
HS_GPAScale	DOUBLE
HS_Rank	INT
HS_GraduationDate	TIMESTAMP
HS_Name	VARCHAR
HS_Code	VARCHAR
HS_City	VARCHAR
HS_State	VARCHAR
HS_Zip	VARCHAR
HS_MnSCURegion	VARCHAR
HS_District	VARCHAR
HS_DistrictCode	VARCHAR
ACTScore	DOUBLE
LibraryUsed	BOOLEAN

Table 7: Majors table in Hive

Field	**Type**
UniqueMajorId (Primary Key)	INT
UniqueId	VARCHAR
Major	VARCHAR
MajorCode	VARCHAR
MajorProgram	VARCHAR
MajorDepartment	VARCHAR
MajorSchool	VARCHAR
MajorCollege	VARCHAR
FY	VARCHAR
UniqueStudentId	INT

Table 8: CirculationLog table in Hive

Field	**Type**
Uniquecirculationid	BIGINT
UniqueId	VARCHAR
YearTerm	VARCHAR
TermName	VARCHAR
Date	VARCHAR
DateOfTerm	INT
Hour	VARCHAR
Action	VARCHAR
Id	VARCHAR
Budget	VARCHAR
Profile-id	VARCHAR
Barcode	VARCHAR
Material	VARCHAR
Item-status	VARCHAR
Collection	VARCHAR
Description	VARCHAR
Doc-title	VARCHAR
UniqueStudentId	INT

Table 9: TwitterAnalysis table in MySQL and Hive

Field	**Type**
UniqueID	BIGINT
TweetID	BIGINT
CreatedAt	VARCHAR
Tweet	VARCHAR
FavouriteCount	BIGINT
ReTweetCount	BIGINT
Lang	VARCHAR
UserID	BIGINT
UserName	VARCHAR
ScreenName	VARCHAR
Location	VARCHAR
FollowersCount	BIGINT
FriendsCount	BIGINT
Statuses	BIGINT
Timezone	VARCHAR

Loading Data from HDFS to Hive Tables

The following command is used to load data from an HDFS file path to Hive database tables.

```
hive> load data inpath '/user/student/Students' into table Students;

hive> load data inpath '/user/student/Majors' into table Majors;

hive> load data inpath '/user/student/CirculationLog' into table CirculationLog;

hive> load data inpath '/user/student/TwitterAnalysis' into table TwitterAnalysis;
```

Chapter 5: Analysis and Results

Access to Hadoop Cluster

To visit the Hadoop Cluster web, use the following link.

http://10.59.7.90:8088/cluster, which was deployed in Business Computing Research Laboratory of Saint Cloud State University. The below screen refers to All Applications available in Hadoop Cluster.

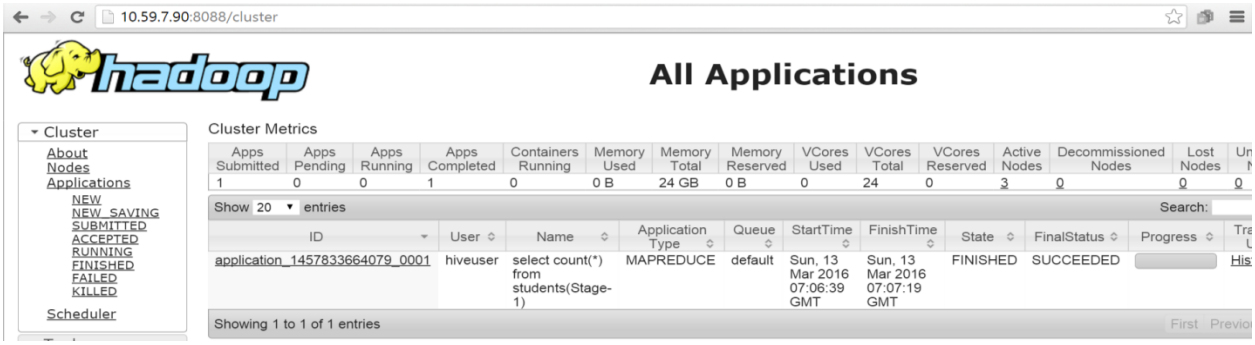


Figure 7: Hadoop Cluster – All Applications

/cluster/nodes points to the Active nodes of Hadoop Cluster, the screen below refers to all 3 active data nodes in the Hadoop Cluster.

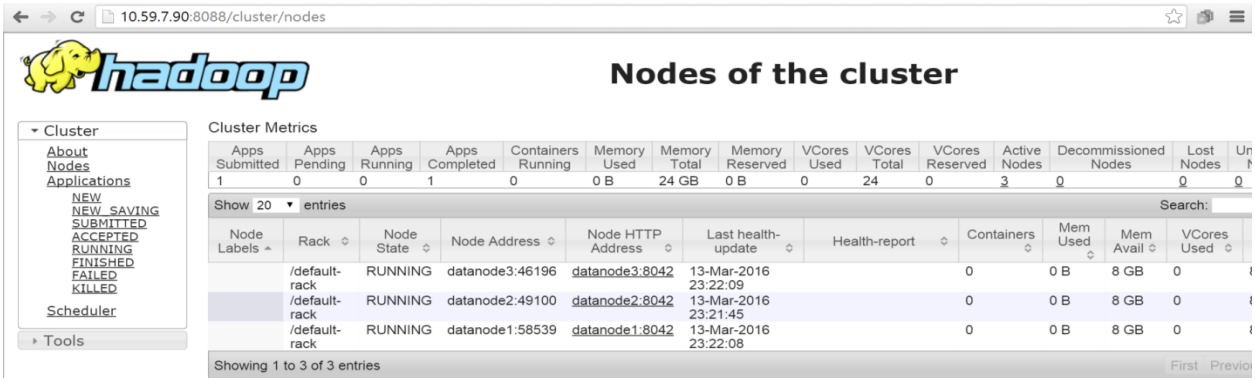


Figure 8: Hadoop Cluster – Active Nodes of the cluster

/cluster/nodes/lost path takes to the Lost nodes of Hadoop Cluster.

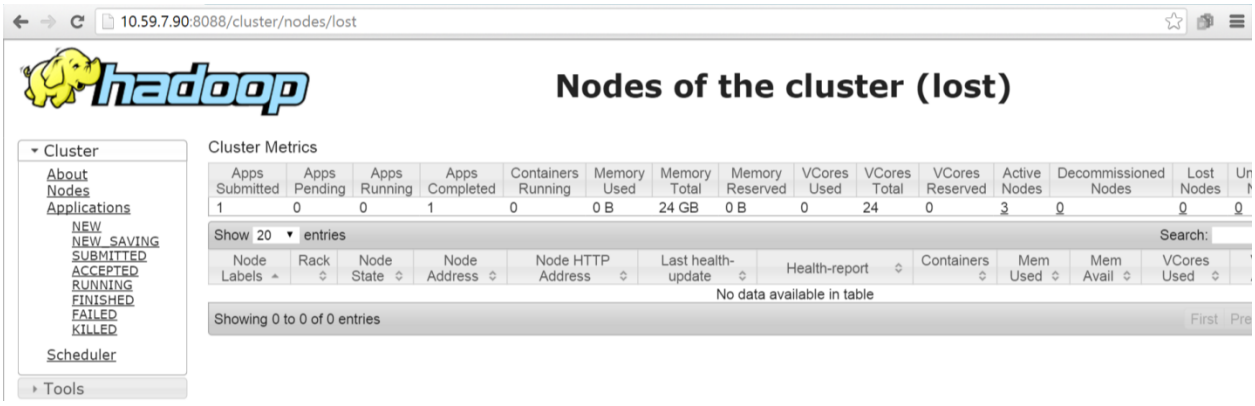


Figure 9: Hadoop Cluster – Lost Nodes of the cluster

/cluster/nodes/unhealthy path takes to the Unhealthy nodes of Hadoop Cluster.

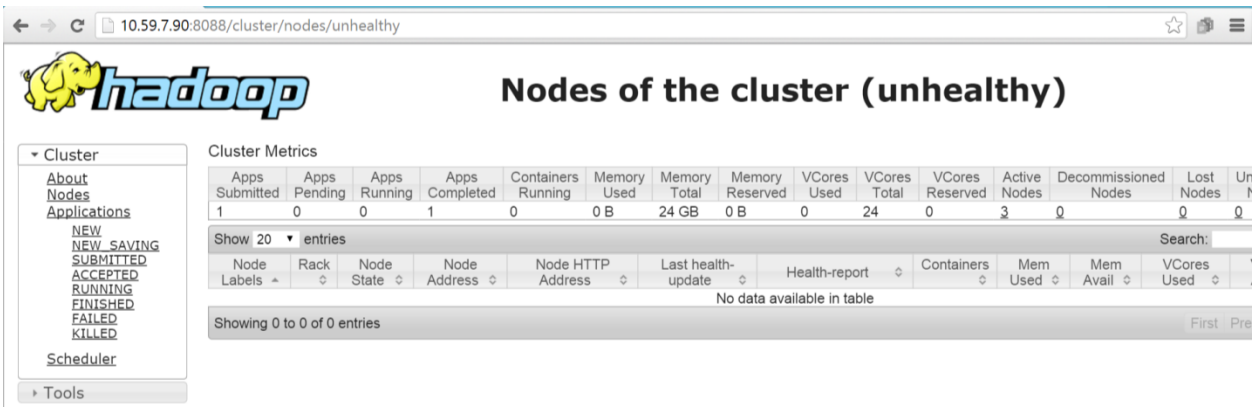


Figure 10: Hadoop Cluster – Unhealthy Nodes of the cluster

/cluster/nodes/decommissioned path takes to the Decommissioned nodes of Hadoop Cluster.

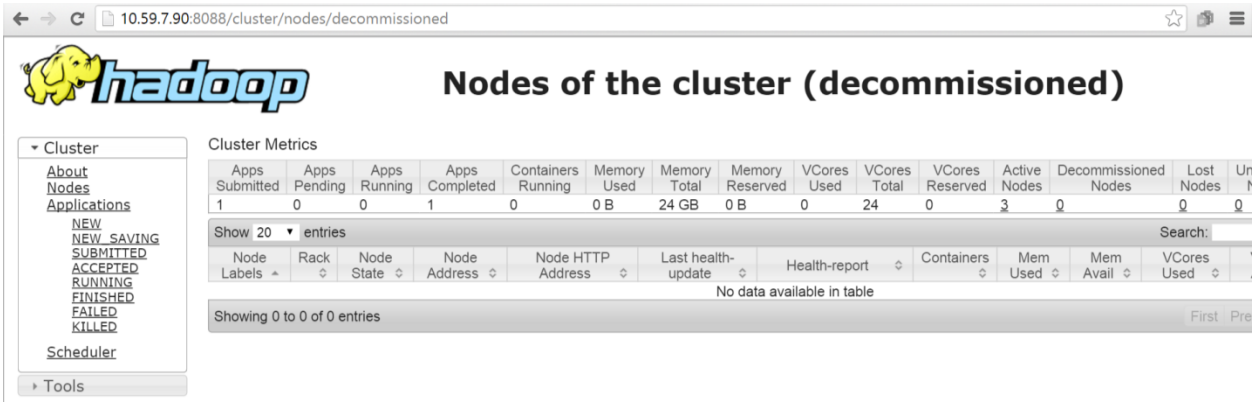


Figure 11: Hadoop Cluster – Decommissioned Nodes of the cluster

/cluster/nodes/rebooted path takes to the Rebooted nodes of Hadoop Cluster.

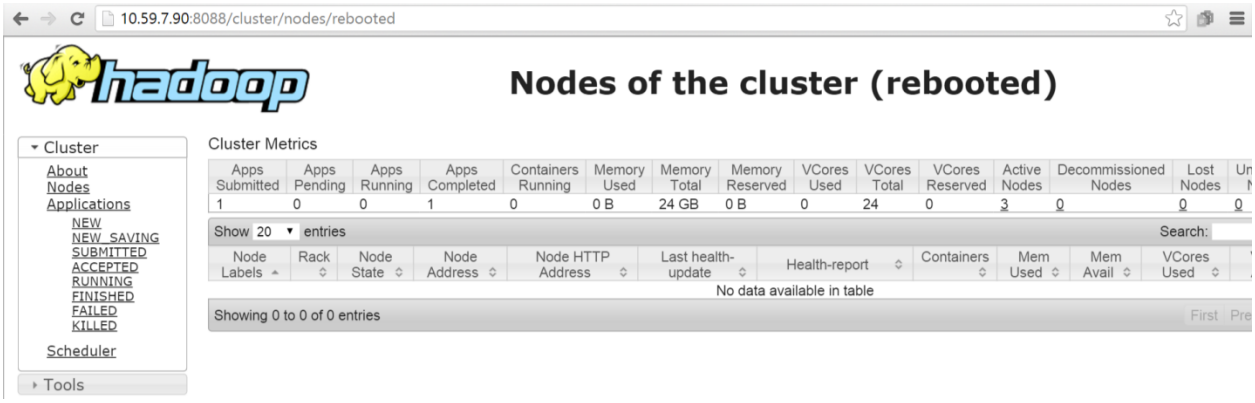


Figure 12: Hadoop Cluster – Rebooted Nodes of the cluster

/cluster/apps path takes to Applications running within the Hadoop Cluster.

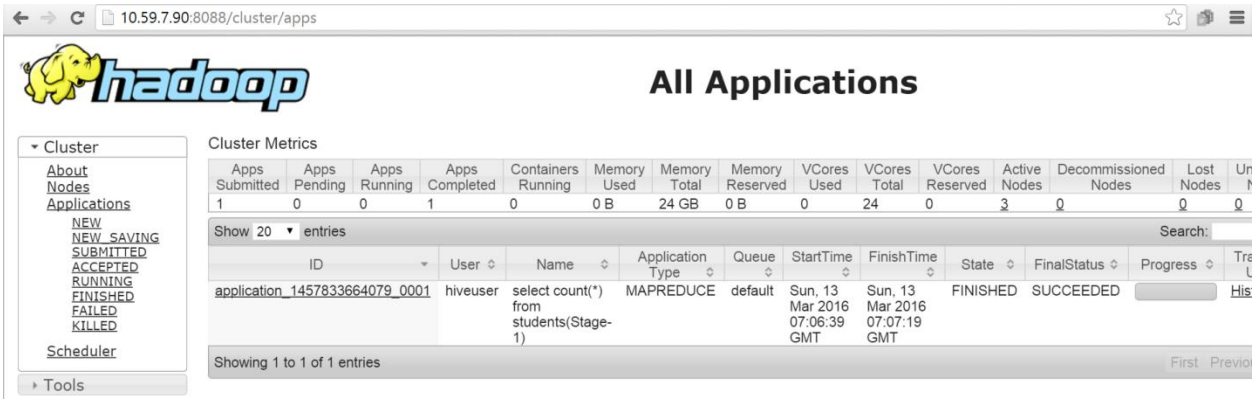


Figure 13: Hadoop Cluster – All Active Applications

/cluster/app/application_<Application_ID> path takes to detailed information about applications that ran within the Hadoop Cluster.

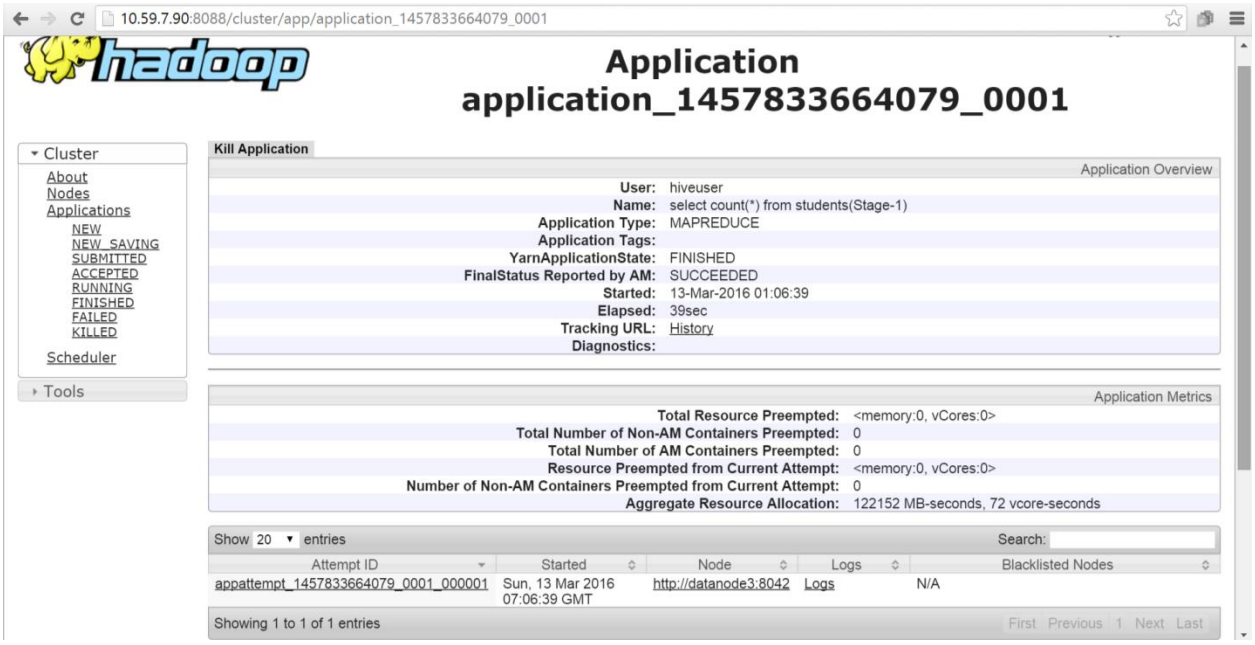


Figure 14: Hadoop Cluster – Application in detail

/conf path allows us to view the configuration of Hadoop Cluster.

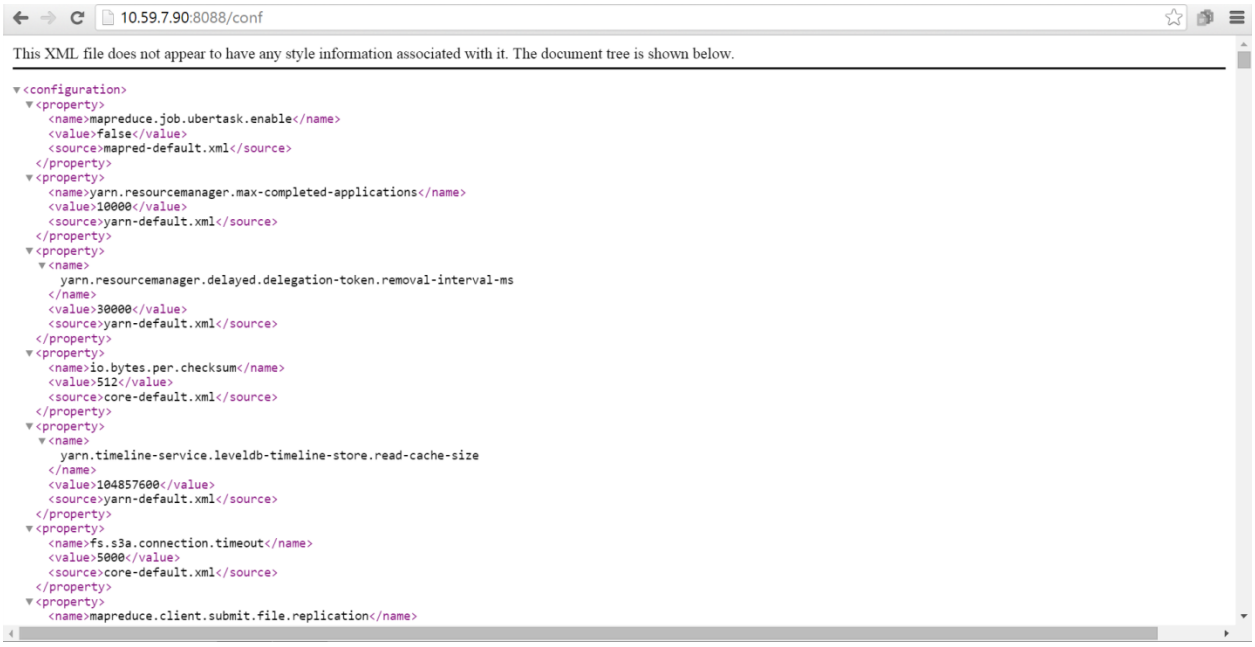


Figure 15: Hadoop Cluster – Current run configurations

/logs path allows us to view the Hadoop Cluster logs, here it displays logs of namenode, secondarynamenode, resourcemanager of masternode.

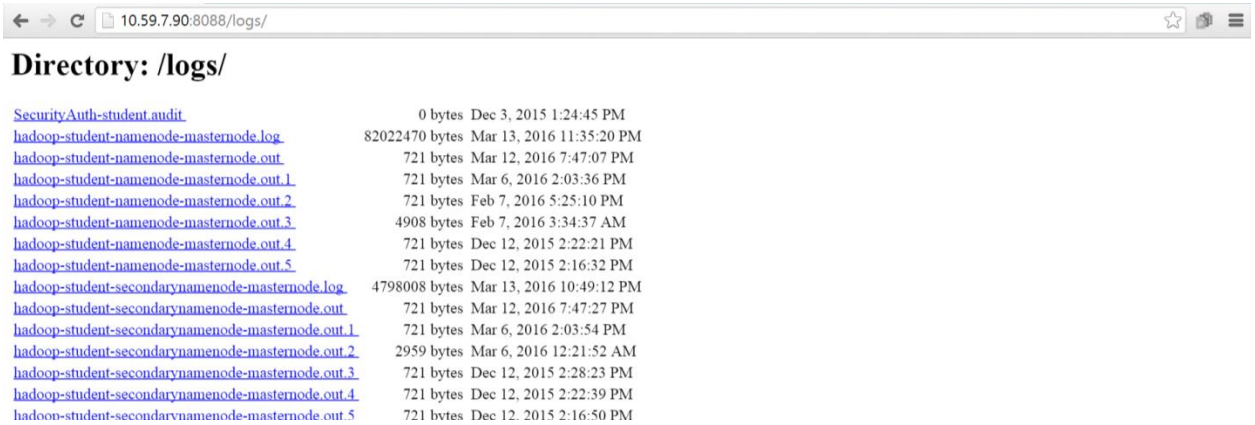


Figure 16: Hadoop Cluster – Logs

http://10.59.7.90:51070 path takes to the Namenode summary information, where it displays the version, deployment date etc.

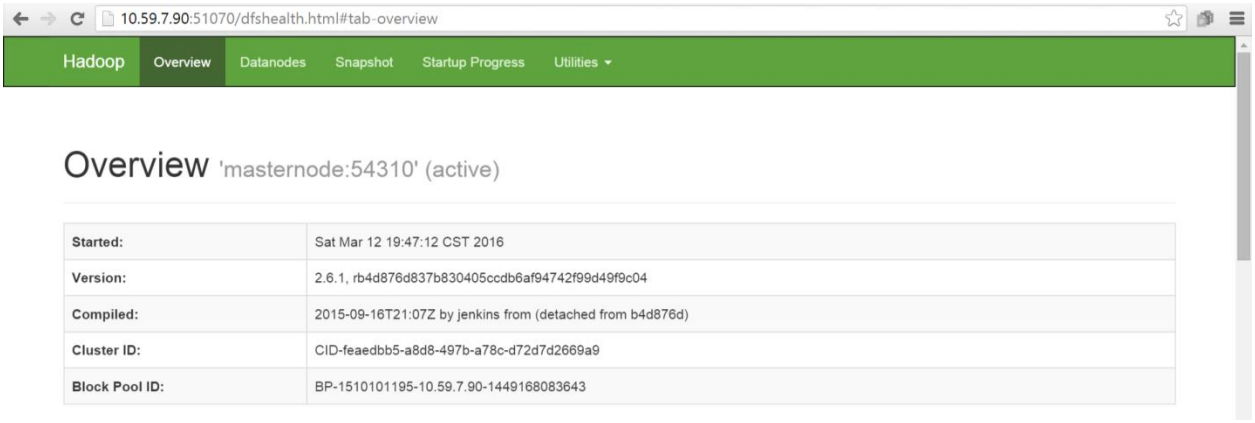


Figure 17: Namenode overview

Here's the summary for the namenode, which shows the server space details, live nodes, dead nodes, decommissioned nodes, storage directory, free space in server, DFS and Non DFS used details etc.

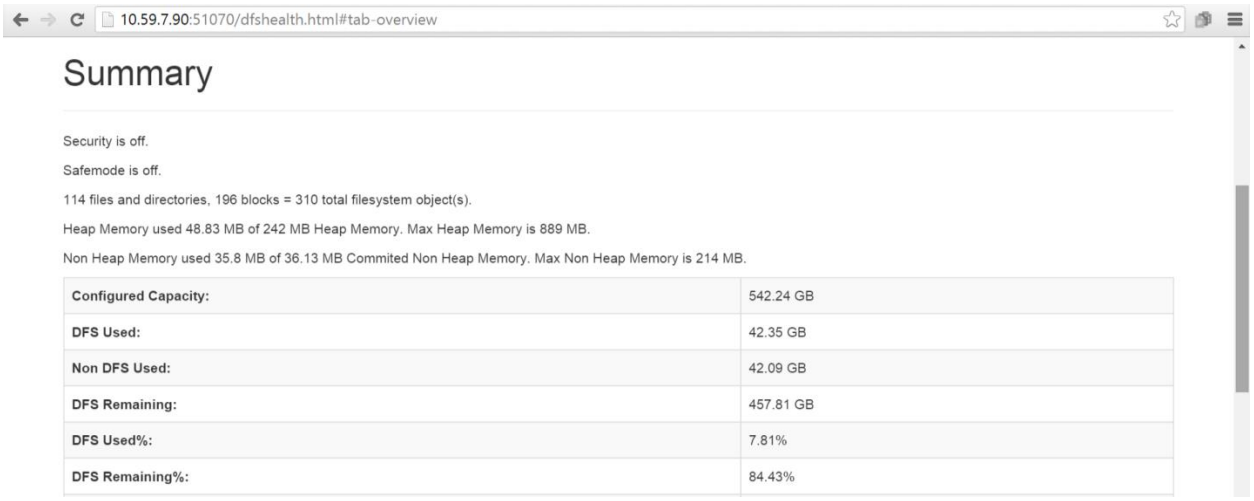


Figure 18: Namenode information

Datanodes tab takes to Datanode Information, where its shows detail of Active datanodes, Decommissioned datanodes.

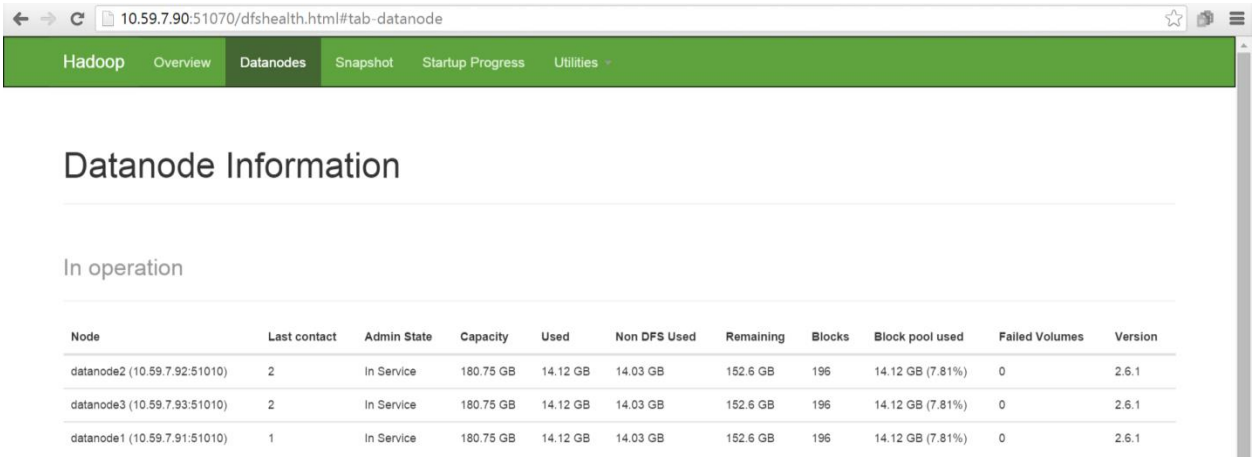


Figure 19: Datanode information

/logs path takes to the log file details, an alternate way to access log files apart from Hadoop Cluster.

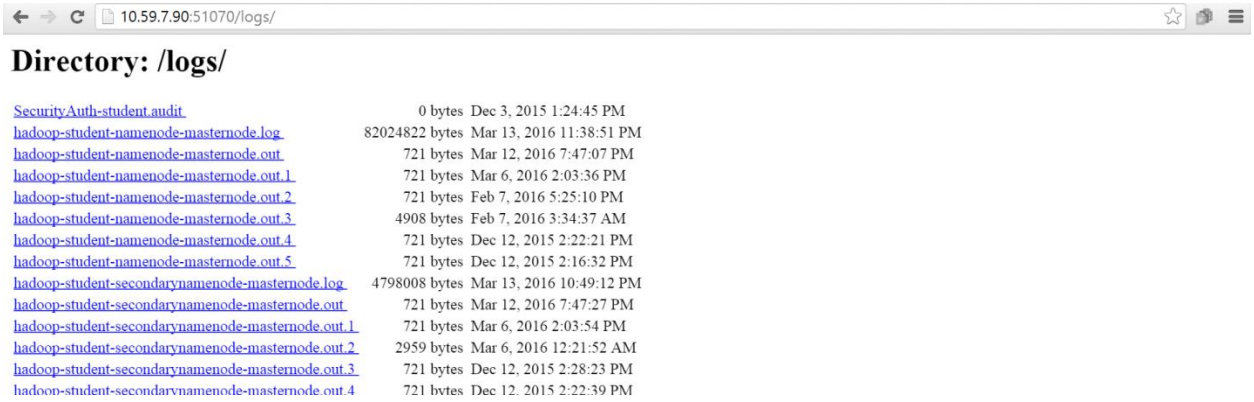


Figure 20: Hadoop cluster logs from Namenode

Utilities tab provides a means to access HDFS FileSystem online, where a full data drill down will take place to retrieve the HDFS data.

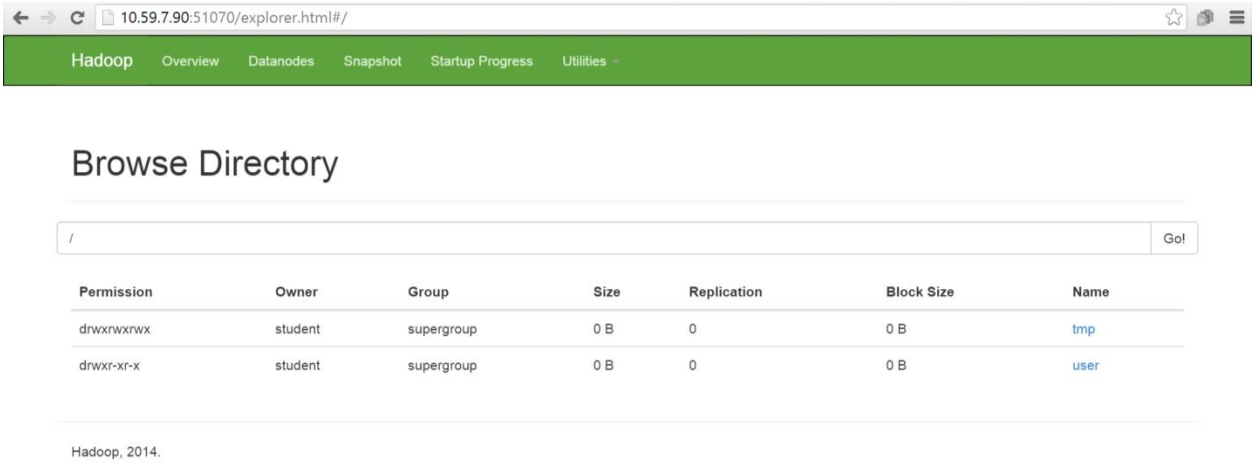


Figure 21: Available HDFS FileSystem data

Access to Apache Tomcat

The http://10.59.7.90:8080 path takes to the Tomcat Homepage, where the Java Web application is deployed.

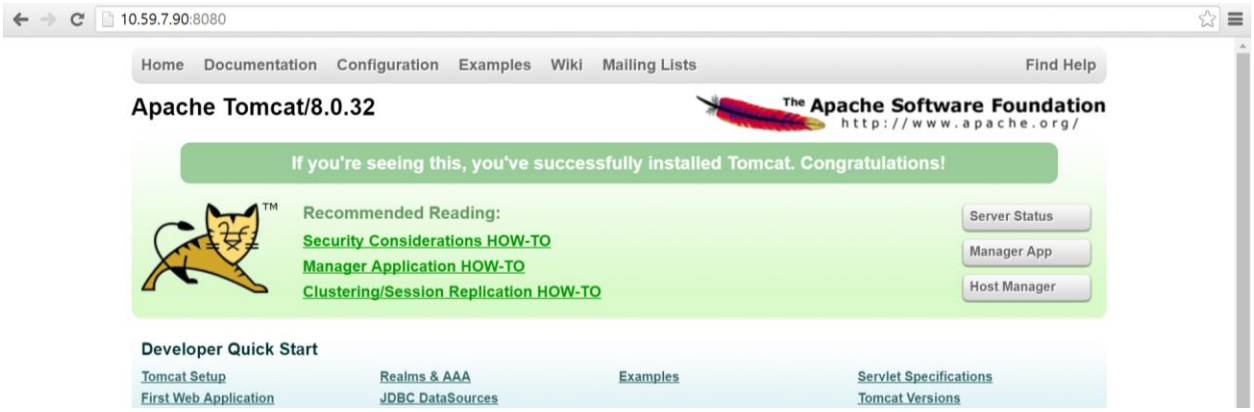


Figure 22: Apache Tomcat Homepage

"Manager App", takes us to the Access Manager App, where the application owner does the application deployment and decommissioning. Login details were configured in tomcat configuration files available at tomcat-users.xml file.

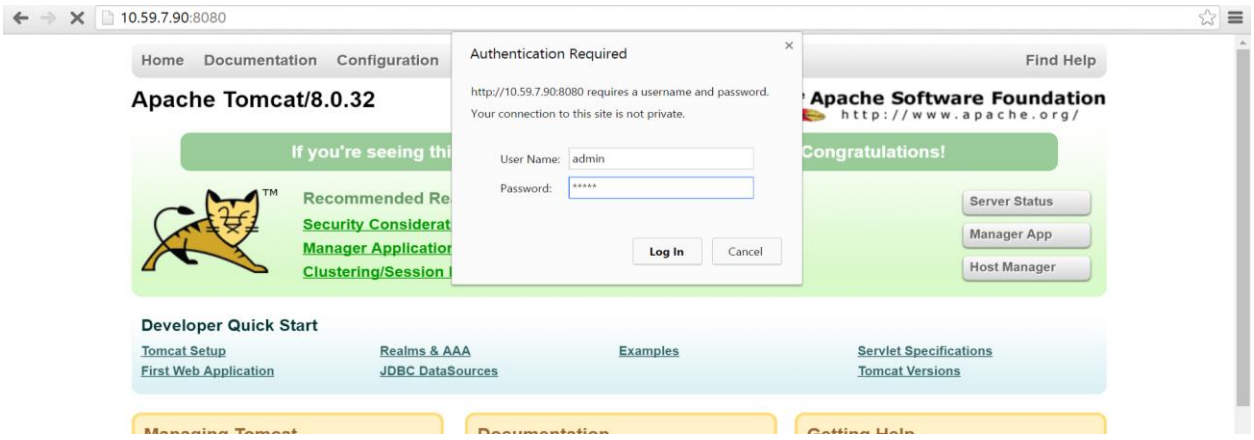


Figure 23: Apache Tomcat Application Manager login

Application source code is deployed using a WAR file, thus the WAR file is used to deploy Hadoop WAR file in the Tomcat Application server by choosing the .war file.

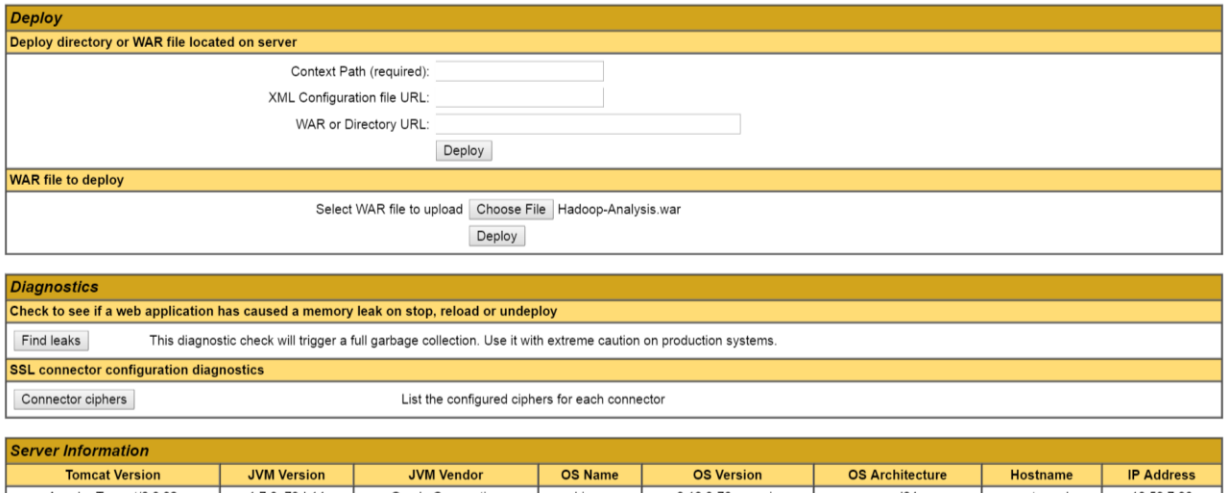


Figure 24: Apache Tomcat WAR file to deploy in Application Manager

After successful deployment, Access the Web application by clicking /Hadoop-Analysis paths available for applications.

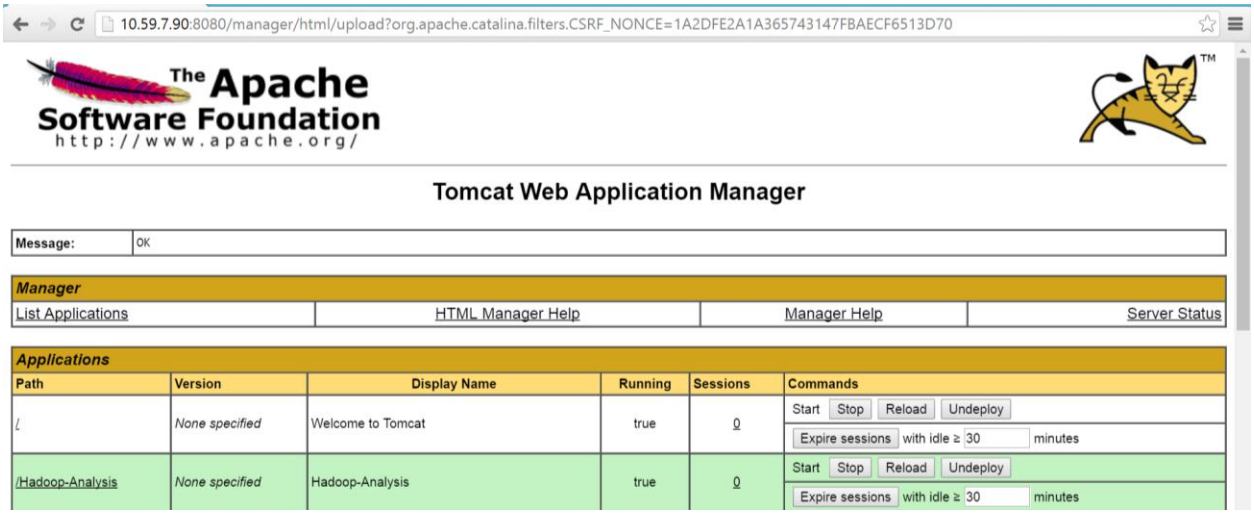


Figure 25: Apache Tomcat Application Manager

Access to Java Web Application

The http://10/59.7.90:8080/Hadoop-Analysis URL takes to Web applications built using Java Server Pages and Java Servlet framework to connect to Hive and MySQL. This code allows us to run queries and to perform analysis on results provided when comparing the performance between Hive and MySQL queries.

Figure 26: Web Application Login Page

A user validation showing the error message “Username and Password are required fields. The blank value submission of Username and Password causes redirection to the Login Page.

Figure 27: Web Application Login Page error for empty submission

User validation showing the error message “Authentication failure” for an invalid login submission of Username and Password and then redirecting to the Login Page.

Figure 28: Web Application Login Page error for authentication failure

On successful Login, it redirects to the Home Page with welcome message. By default, it navigates to a MySQL query processor.

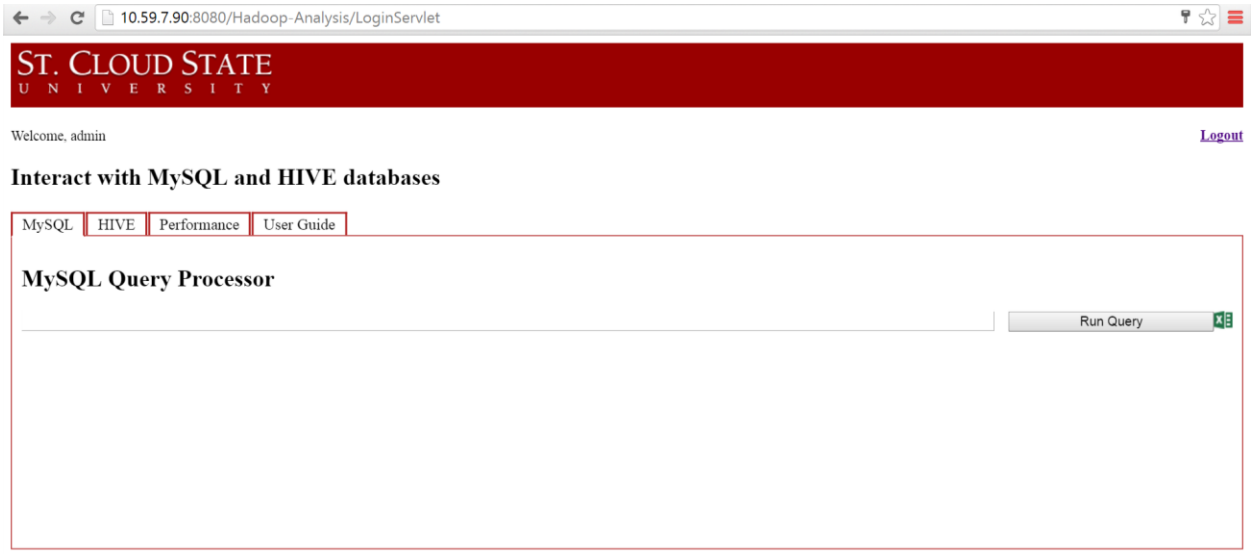


Figure 29: Web Application Home Page as MySQL Query Processor

On invalid query submission, it displays error message “Invalid MySQL Query”, Query must follow MySQL query standards.

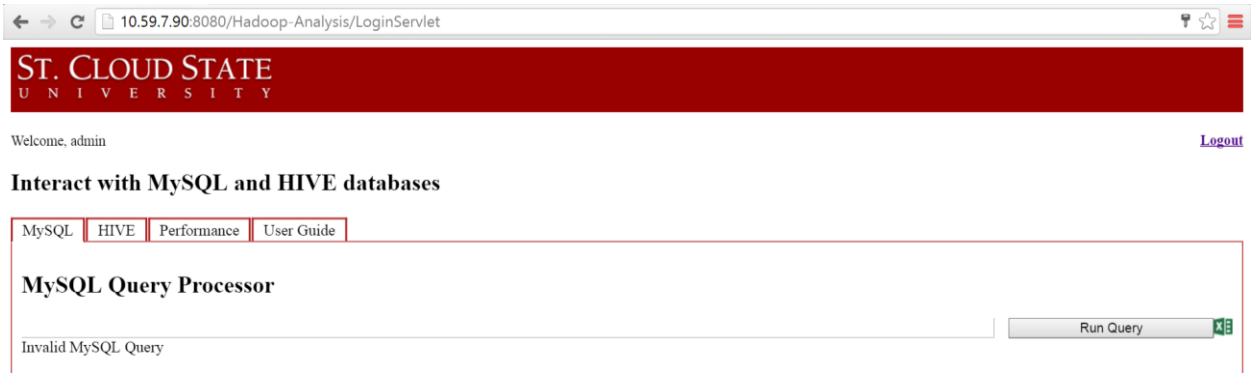
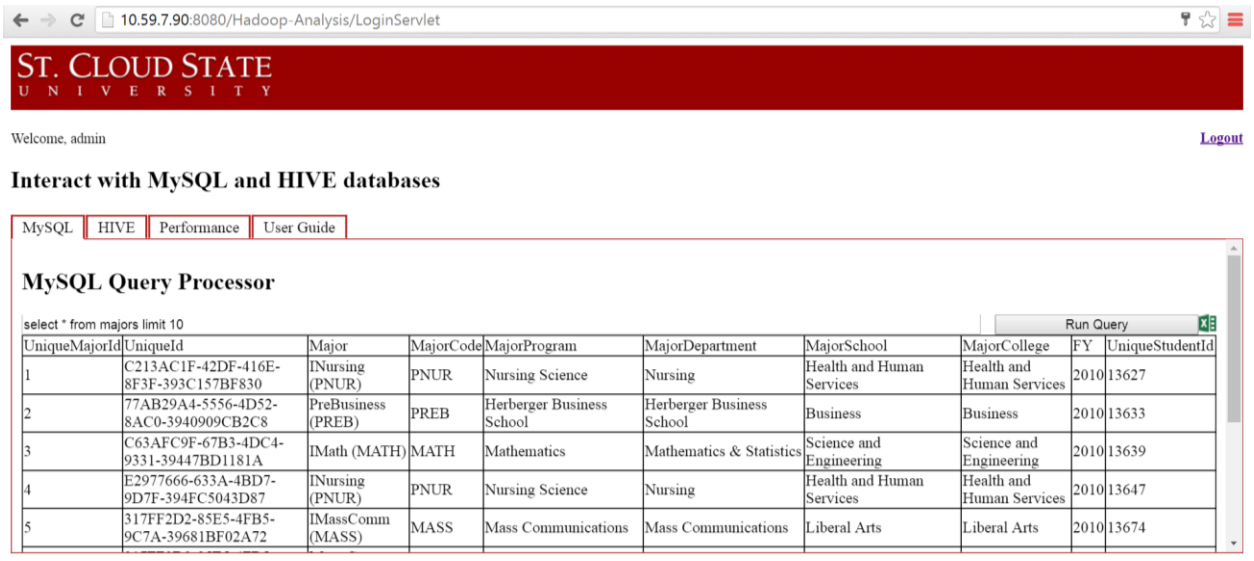


Figure 30: Web Application error for invalid MySQL query

Sample query to see the results of first 10 records of Majors table in the MySQL database. Because the user is logged in as admin, it grants special access to export results in Microsoft Excel file.



UniqueMajorId	UniqueId	Major	MajorCode	MajorProgram	MajorDepartment	MajorSchool	MajorCollege	FY	UniqueStudentId
1	C213AC1F-42DF-416E-8F3F-393C157BF830	INursing (PNUR)	PNUR	Nursing Science	Nursing	Health and Human Services	Health and Human Services	2010	13627
2	77AB29A4-5556-4D52-8AC0-3940909CB2C8	PreBusiness (PREB)	PREB	Herberger Business School	Herberger Business School	Business	Business	2010	13633
3	C63AFC9F-67B3-4DC4-9331-39447BD1181A	IMath (MATH)	MATH	Mathematics	Mathematics & Statistics	Science and Engineering	Science and Engineering	2010	13639
4	E2977666-633A-4BD7-9D7F-394FC5043D87	INursing (PNUR)	PNUR	Nursing Science	Nursing	Health and Human Services	Health and Human Services	2010	13647
5	317FF2D2-85E5-4FB5-9C7A-39681BF02A72	IMassComm (MASS)	MASS	Mass Communications	Mass Communications	Liberal Arts	Liberal Arts	2010	13674

Figure 31: Web Application results for valid MySQL query

Hive tab navigates to Hive query processor

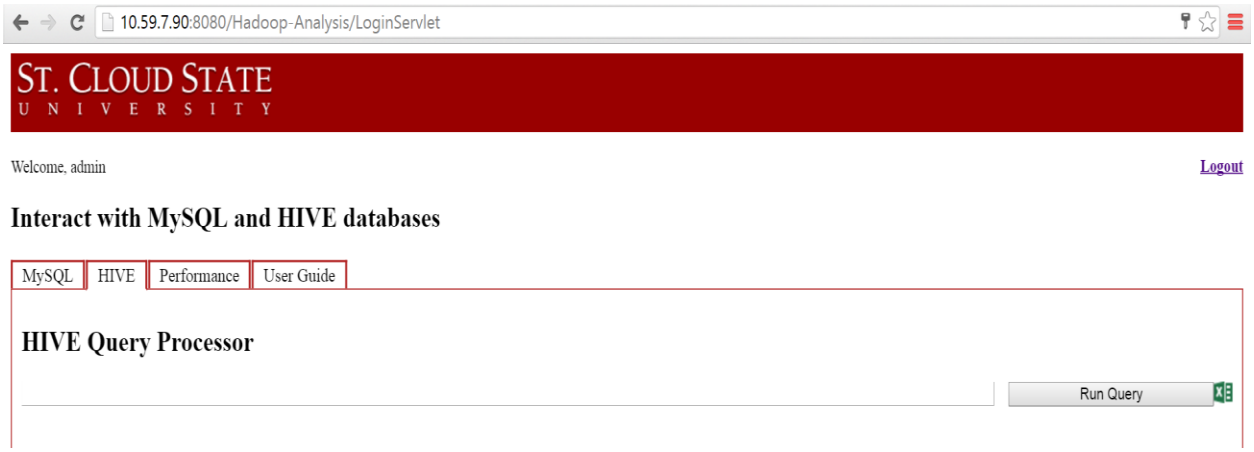


Figure 32: Web Application Hive Query Processor

On invalid query submission, it displays error message “Invalid Hive Query”, the Query must follow HIVESQL query standards.

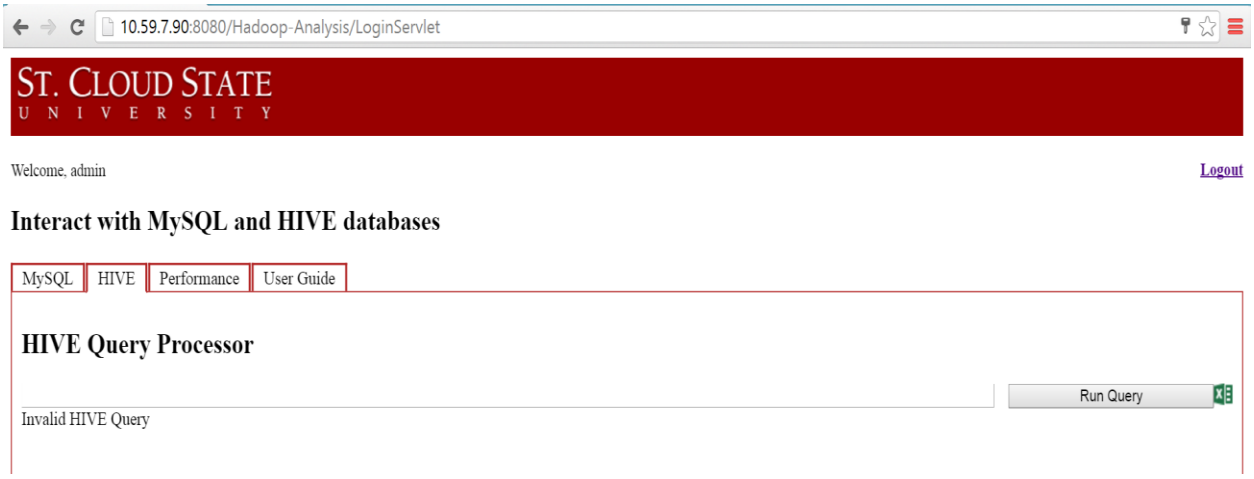


Figure 33: Web Application error for invalid Hive query

A background job will run when a hive query is executed, background processing is handled by the Hive server that runs in the Hadoop Cluster. For every Hive query, a job is triggered by the Hive server to fetch the results.

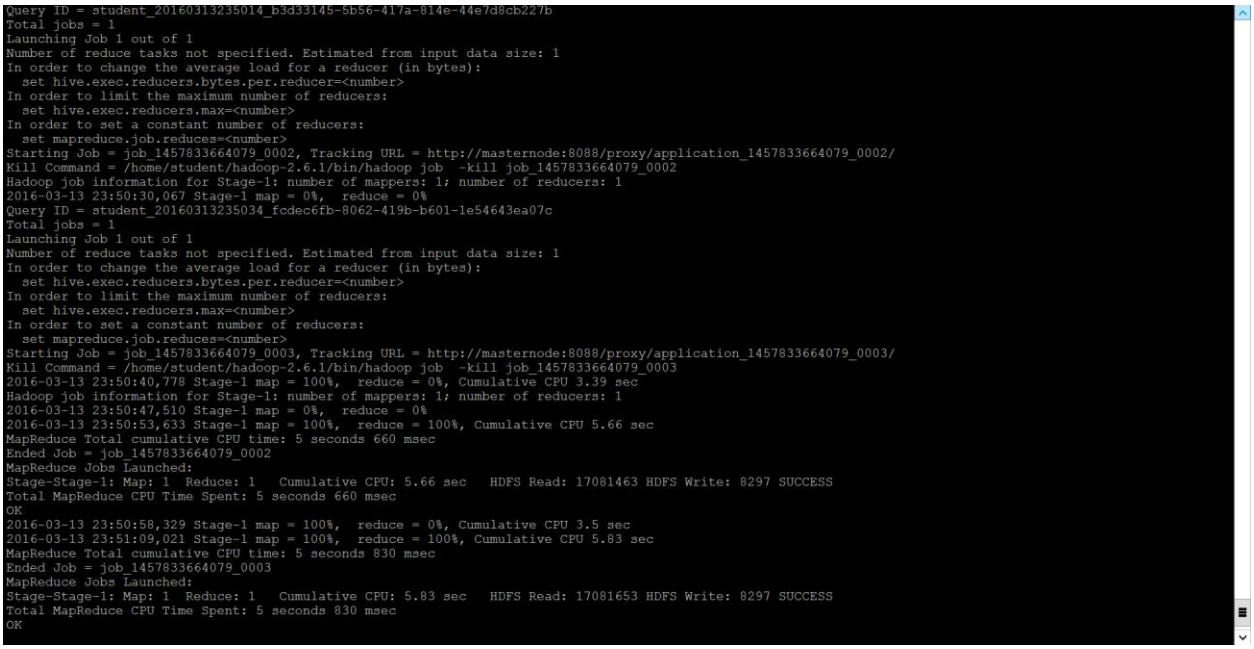


Figure 34: Hive query processing in Hadoop

Sample query to see results of first 10 records of Majors table in Hive database. As the user is logged in as admin, it grants special access to export results in Microsoft Excel file.

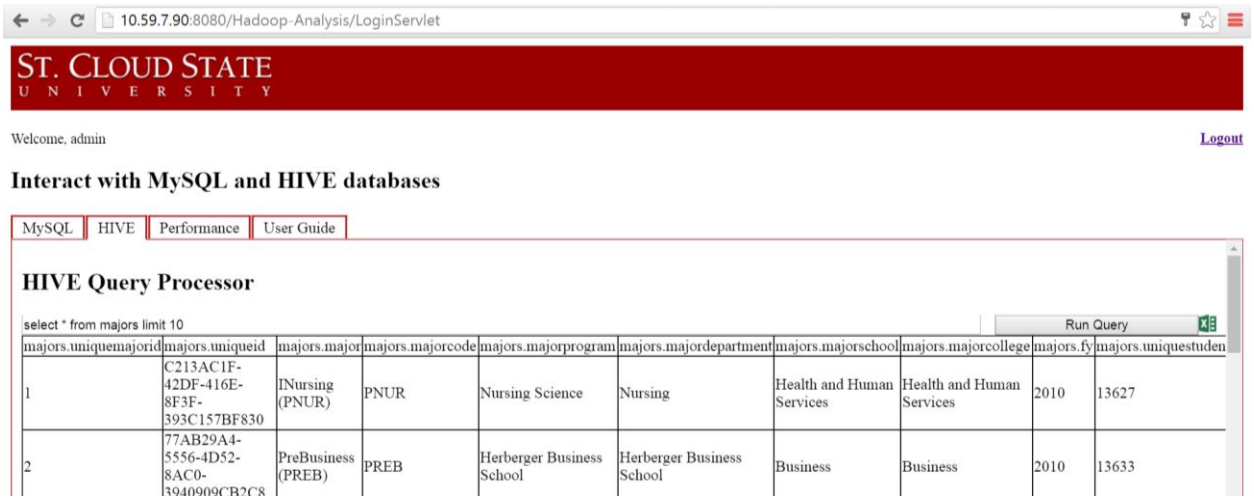


Figure 35: Web Application results for valid Hive query

The Performance Tab shows the time comparison for execution of queries in MySQL and Hive. Because the user is logged in as admin, it grants special access to export the results to a Microsoft Excel file.

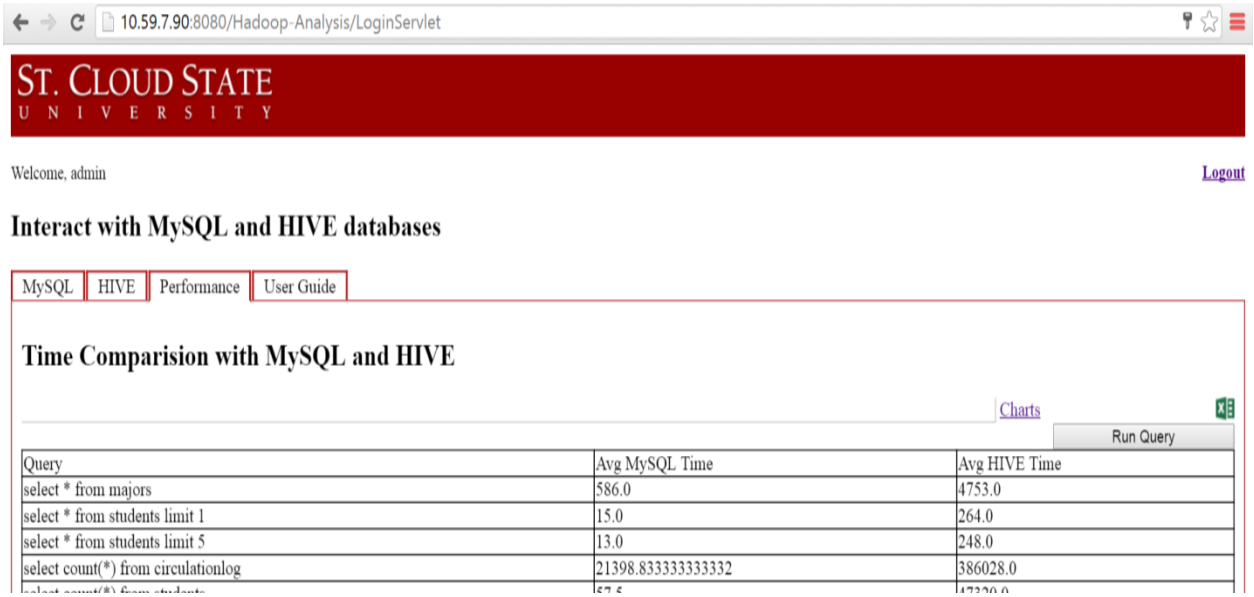


Figure 36: Web Application Time Comparison with MySQL and Hive

The chart below depicts the performance comparison of MySQL and Hive. The X-axis represents the queries while the Y-axis is the time it takes a query to complete its execution in Milliseconds.

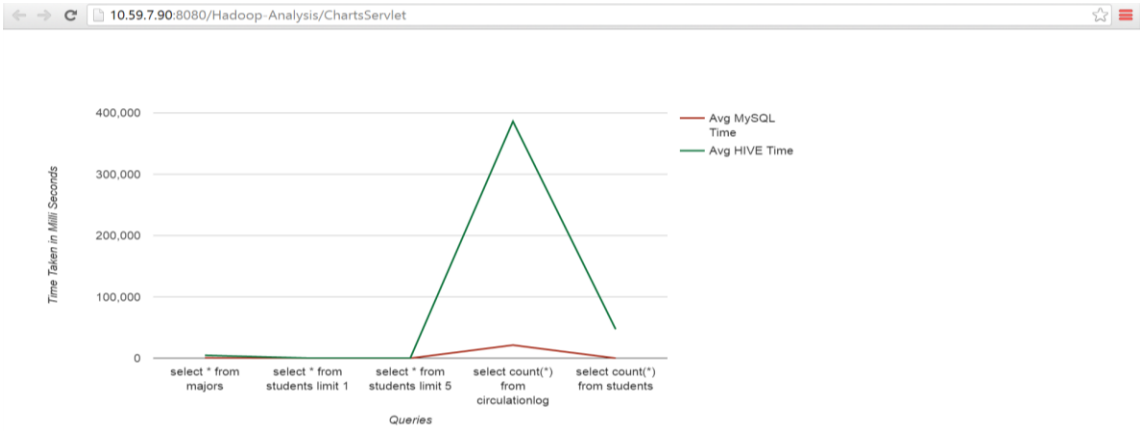


Figure 37: Web Application Time Comparison with MySQL and Hive in Line Chart

The actual values from the comparison of MySQL and Hive data stores appear below.

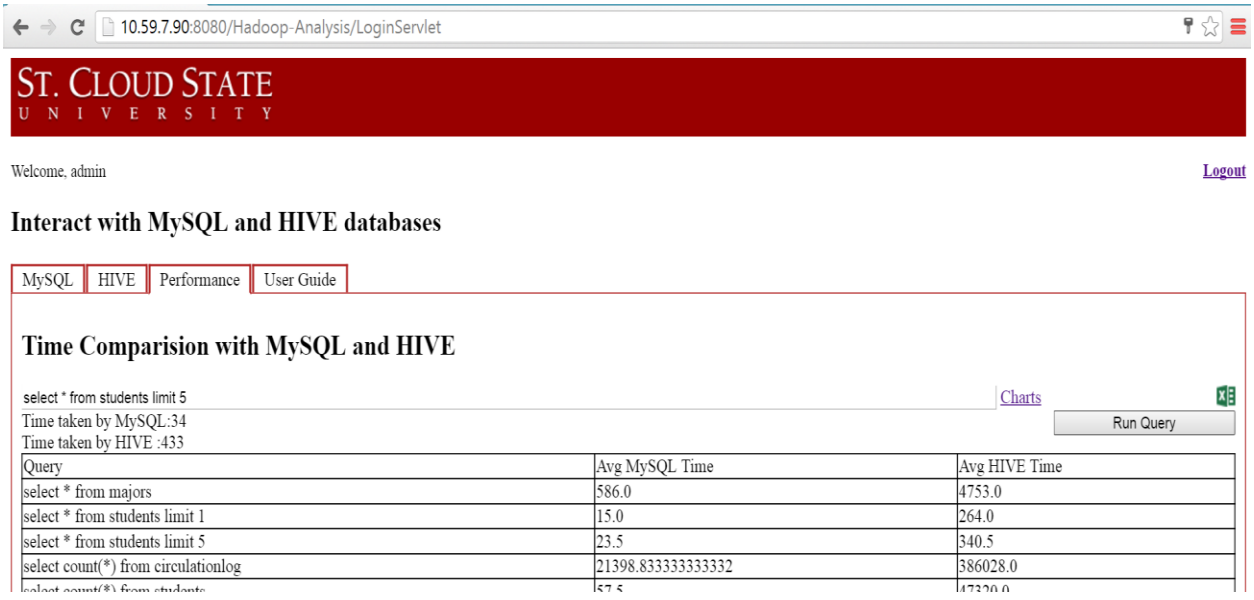


Figure 38: Web Application Time Comparison with MySQL and Hive for given query

The User Guide Tab by default shows MySQL database tables. This page helps users to see a list of tables available in databases.

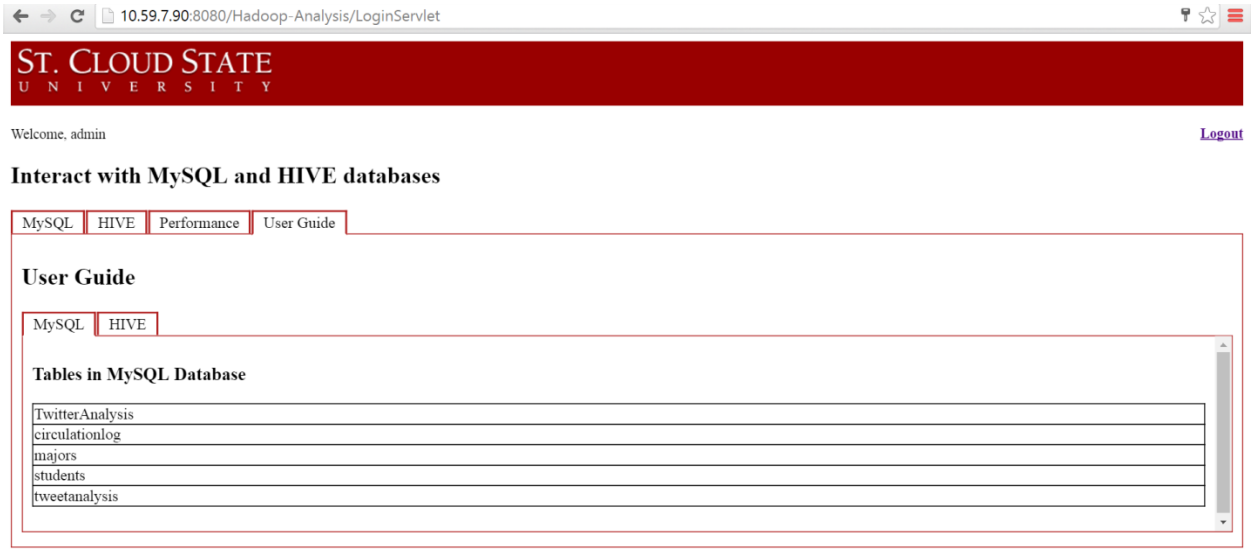


Figure 39: Web Application showing list of tables in MySQL

Click on Hive, to navigate to Hive database tables.

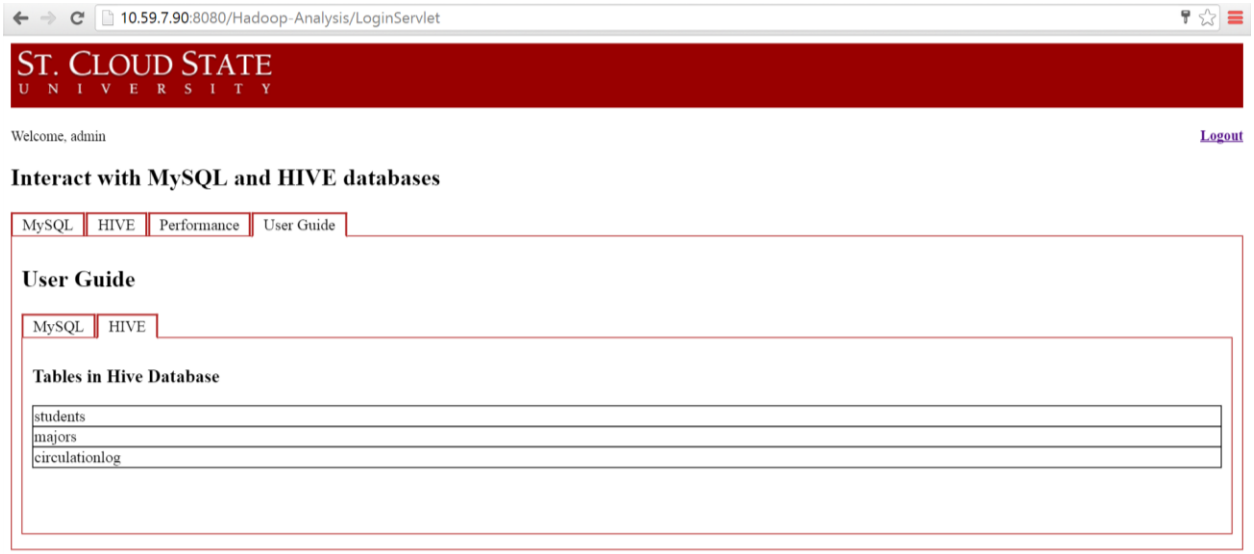


Figure 40: Web Application showing list of tables in Hive

By clicking on the database table in Hive the database table definition appears. Thus, it helps users to identify a list of columns available for each table.

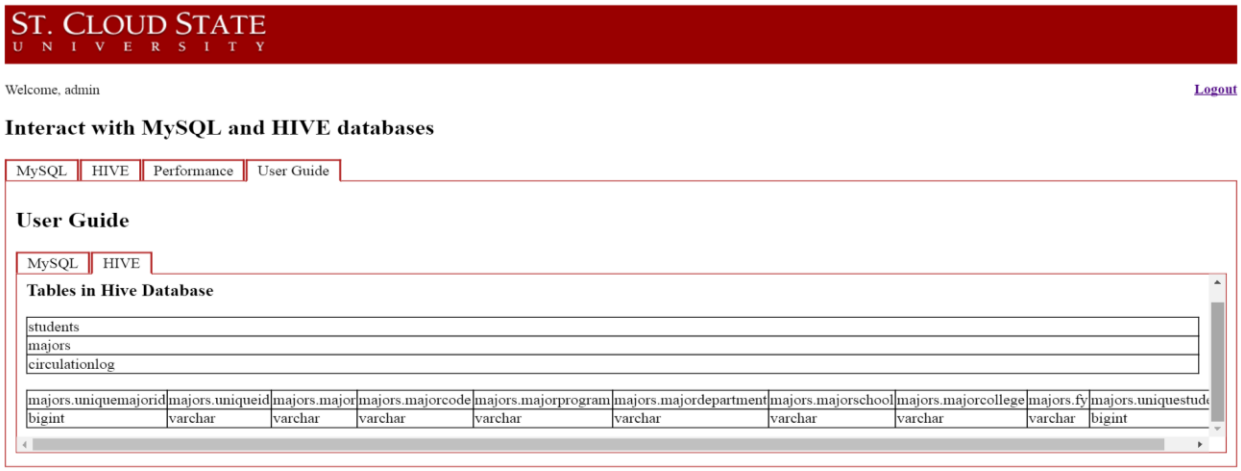


Figure 41: Web Application describing Hive table

Similarly, by clicking on database table in MySQL the database table definition becomes available. Thus, it helps users to identify list of columns available for each table.

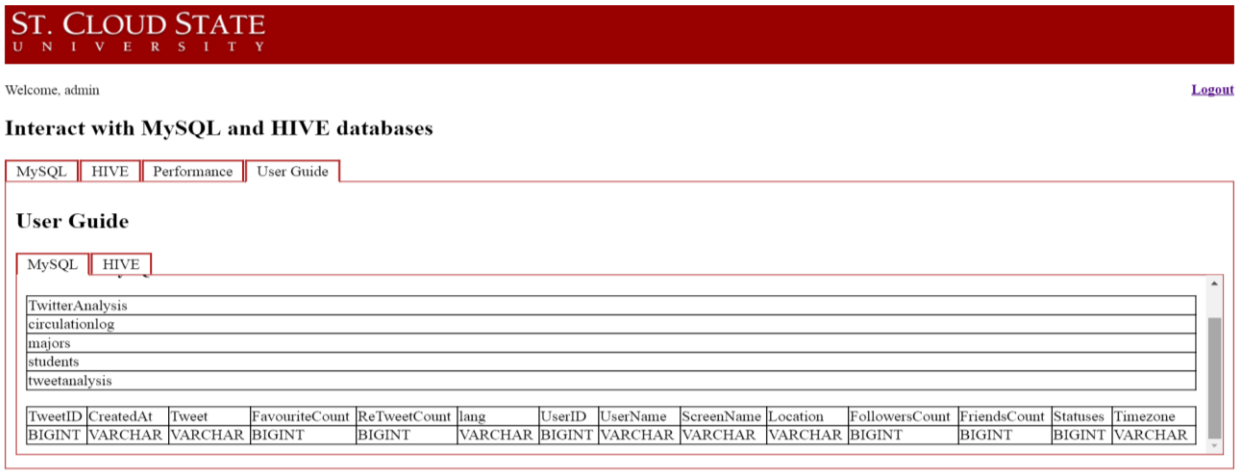


Figure 42: Web Application describing MySQL table

On Logout, it navigates to the Login Page. Based on the user selection of “Remember”, browser cookies then save username and password.

Figure 43: Web Application describing cookie usage

UML Diagrams

The User Login Sequence diagram validates the user login details, fetches user permissions for the webpage, documents performance comparison details, and depicts the database table details for the MySQL and Hive databases.

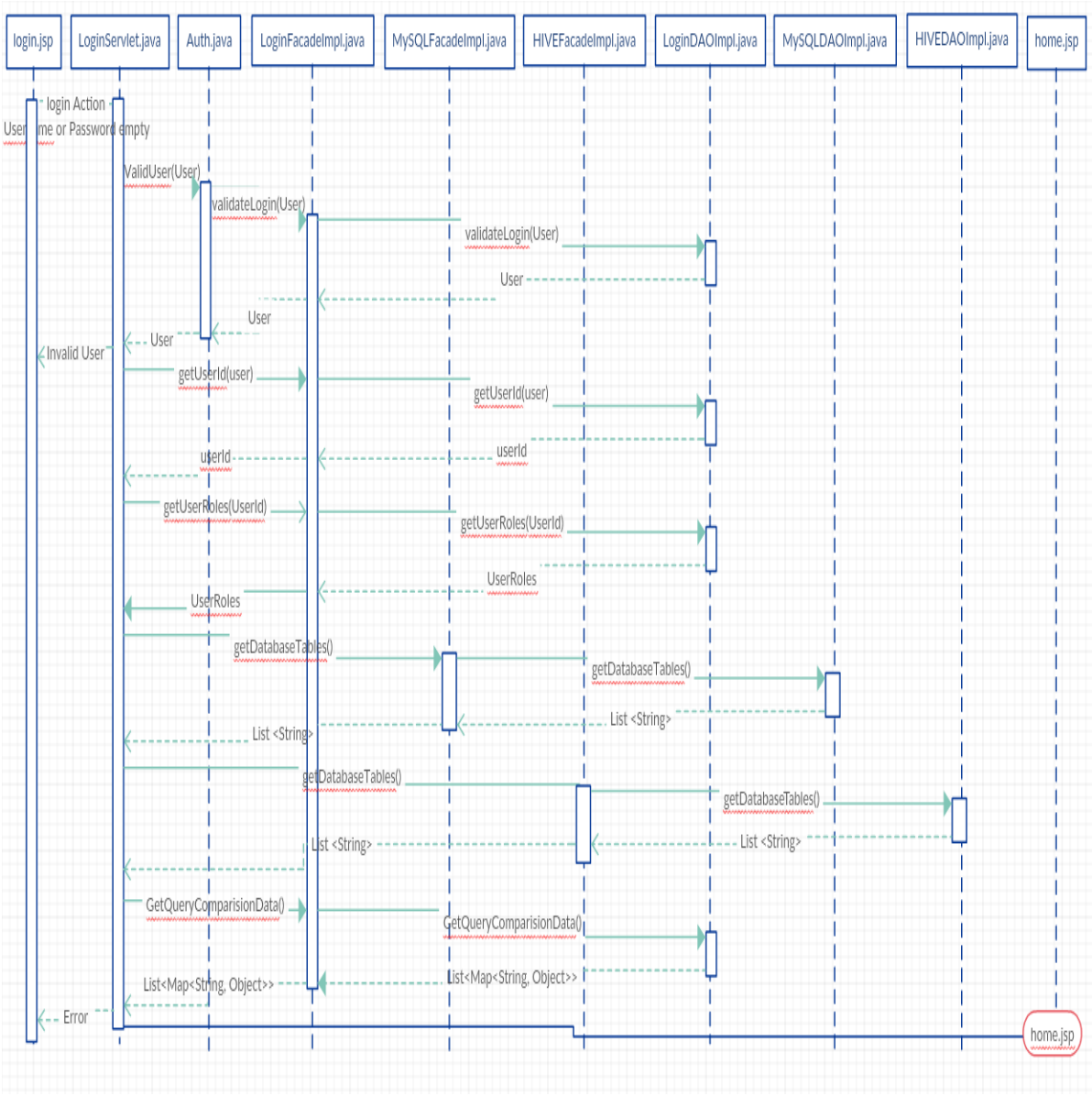


Figure 44: User Login Sequence diagram

Executing a MySQL query from MySQL tab in Home Page Sequence diagram.

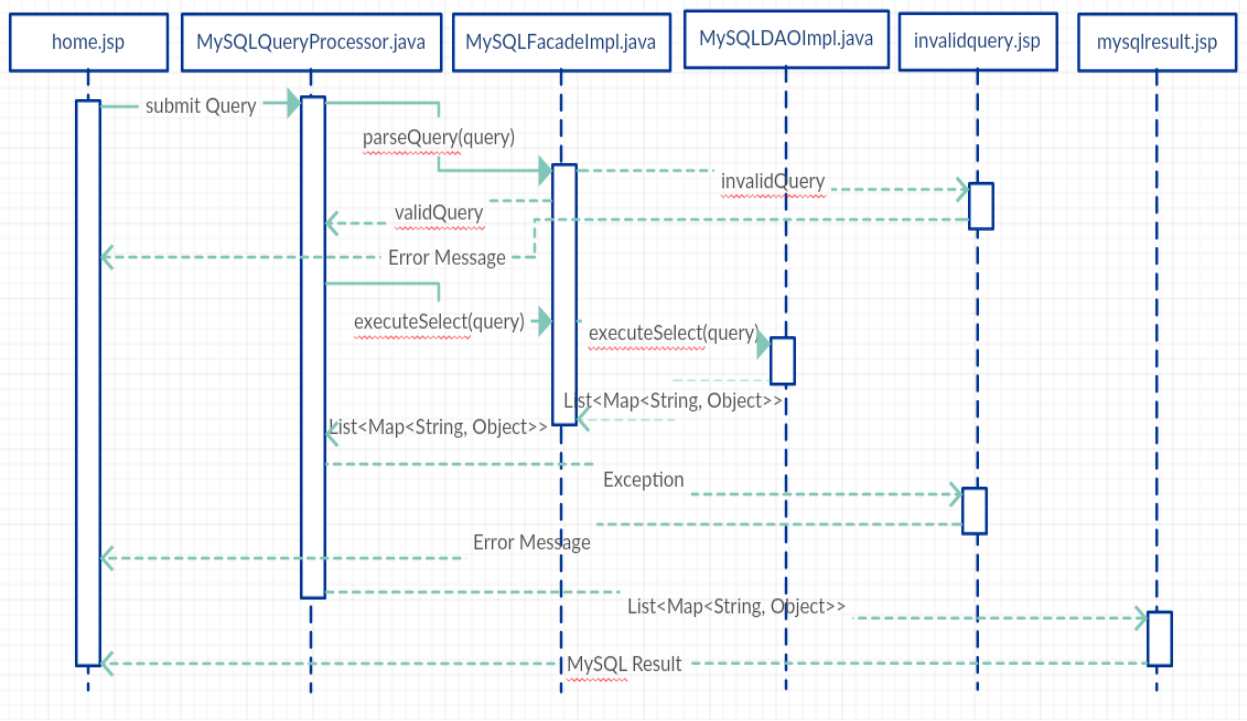


Figure 45: MySQL Query Processor Sequence diagram

Executing a Hive query from Hive tab in the Home Page Sequence diagram.

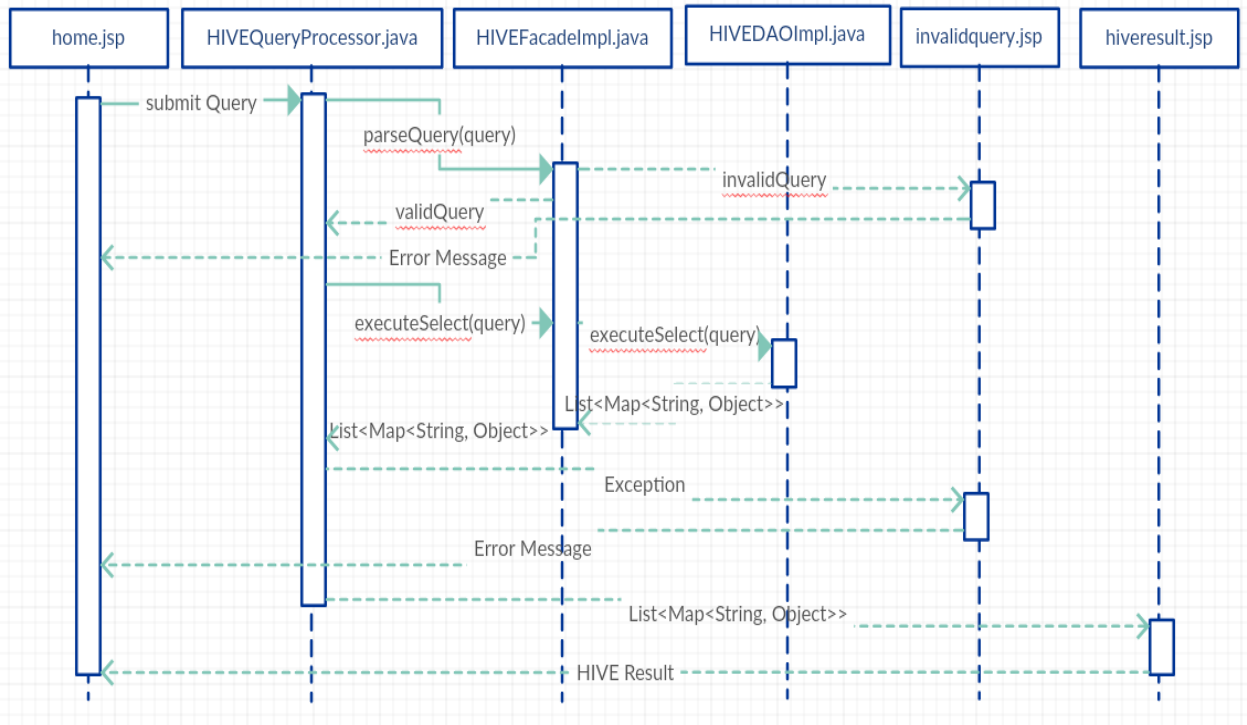


Figure 46: Hive Query Processor Sequence diagram

Display Performance results in Line Chart Sequence diagram.

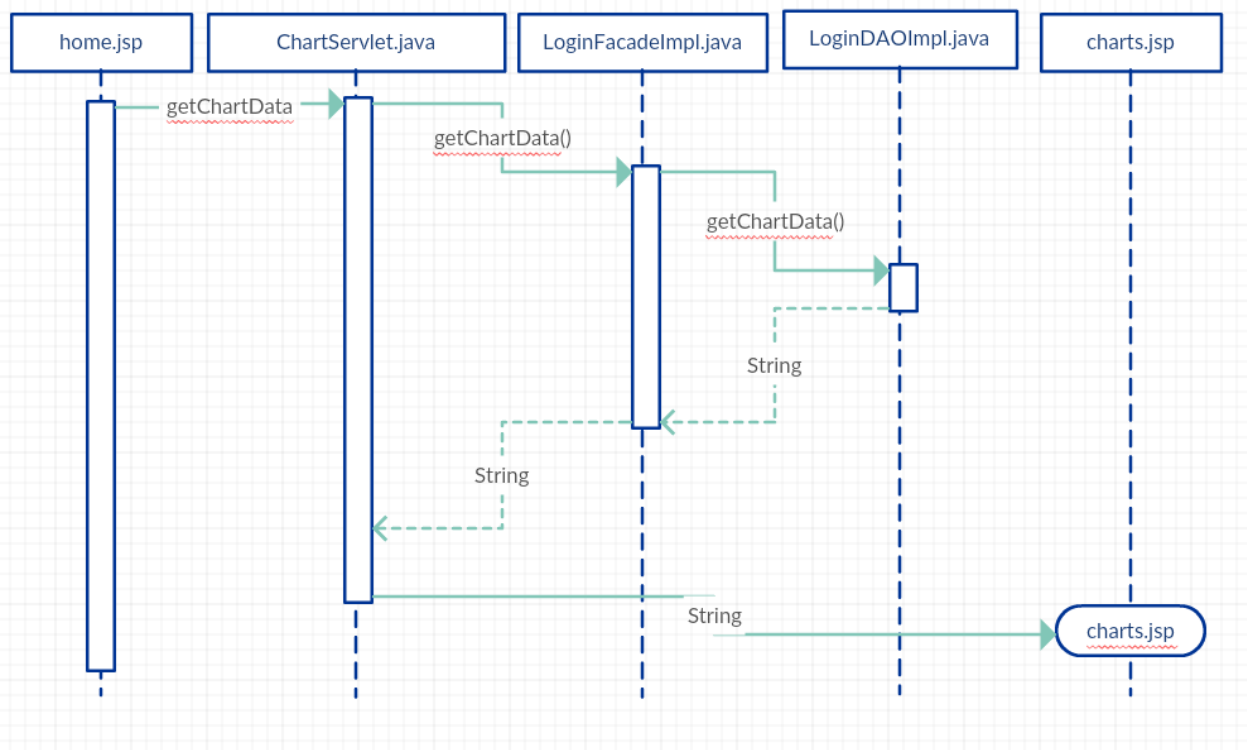


Figure 47: Line Chart Sequence diagram

Summary

The goal of this paper was assess performance using similar data sets stored under two different structures it was important to be able to transfer the exact data between two different data structures. A tool called Sqoop was used to solve this problem and Also provide web interface to data stored in a distributed file system.

Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured data-stores such as relational databases. Its use is critical for this project because the original data being utilized was stored in a MySQL data base. Specifically, this data came

from the circulation log of the campus library and consisted of about 20 Million records and took up about 4GB of data storage. It was transferred into a MySQL database and then Sqoop was used to place it into Hadoop. The structure of the data is defined below via the Hive create table process, which is similar to the same process using basic SQL. Three tables are created: Students, Majors and CirculationLog. Note that in each case a UniqueStudentId is created so that records across the tables can be associated.

A drawing that depicts the process that was followed to undertake the experimental comparison appears below. Both the MySQL database and the HDFS were run on similar hardware within the same cloud. However, the HDFS system was distributed across several nodes. As would be expected the HDFS system performed better in all of the experimental trials.

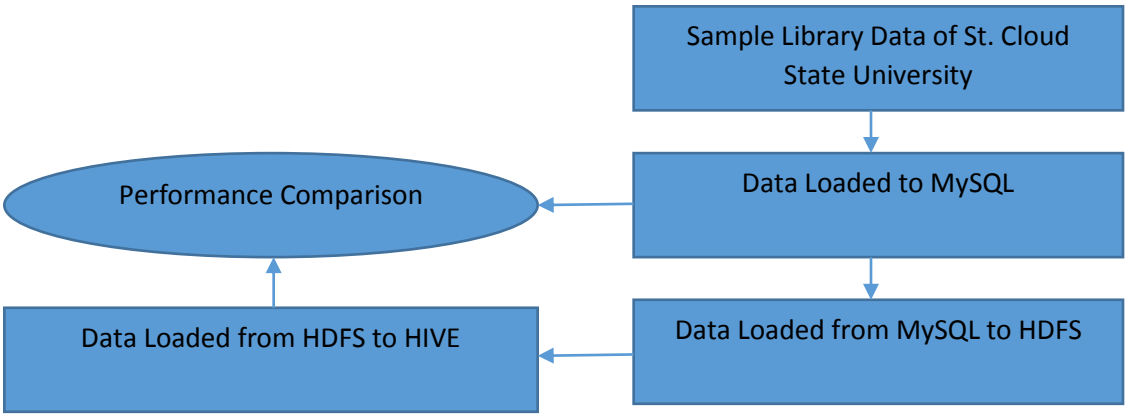


Figure 48: Architecture for MySQL and Hive Performance Comparison

A table of results for the experimental trials appears below. In both cases SQL like code was used to define the query.

Table 10: Comparison of Computation time of Hive vs MySQL.

Query	Hive Computation Time	MySQL Computation Time
Select * from CirculationLog;	7min 32sec	11min 53sec
Select count(*) from CirculationLog;	1min 53sec	2min 35sec
Select count(Distinct UniqueId) from CirculationLog;	58sec	60sec
Select count(*) from CirculationLog c, Students s where s.UniqueId=c.UniqueId;	4min22sec	MemoryException

Chapter 6: Conclusion and Future Work

Conclusion

In this paper, Java web application is developed using Java Server Pages to interact with both traditional databases like MySQL and Hadoop database like Hive in parallel. Also, a comparison of the time taken to execute select queries within MySQL and Hive data stores was carried out. According to the analysis of query execution time, the Hadoop database is preferred when using large datasets while MySQL is preferred when working on small datasets. Java web applications provide an effective and secure platform for users to execute queries and export the results based on user access level.

Future Work

The architecture used herein is flexible and it would be easy to expand the Java web applications devised herein. Customization can also be done on the Hadoop configuration level or Hive Configuration level or at Web services level. Adding improvements in security and performance is always possible by migrating to the latest technologies in web application development. The Java web platform acts as means to expand the scope of the application. This can be done by linking it to software components such as PIG scripting, R Connectors for Statistics, Mongo Databases and much more.

References

A Quick Guide to Using the MySQL APT Repository. Retrieved March 04, 2016, from http://downloads.mysql.com/docs/mysql-apt-repo-quick-guide-en.pdf

Afreen, S. (2016). Using Hadoop to support Big Data Analysis: Design and Performance Characteristics, Working Paper: Information Systems Department, Saint Cloud State University.

Apache Hive - Apache Software Foundation. (n.d.). Retrieved March 12, 2016, from https://cwiki.apache.org/confluence/display/Hive/Home

Apache Tomcat 8. (n.d.). Retrieved March 07, 2016, from http://tomcat.apache.org/tomcat-8.0-doc/introduction.html

Cecchinel, C., Jimenez, M., Mosser, S., & Riveill, M. (2014, June). An architecture to support the collection of big data in the internet of things. In 2014 IEEE World Congress on Services (pp. 442-449). IEEE.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2), 4.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

Dean, J., & Ghemawat, S. (2010). MapReduce: a flexible data processing tool. Communications of the ACM, 53(1), 72-77.

Fleming, D. (2004). Network response time for efficient interactive use. Proceedings of the 20th Computer Science Seminar, Addendum-T2-1. RIP, Hartford Campus, April, 24.

Goth, G. (2015). Bringing big data to the big tent. Communications of the ACM, 58(7), 17-19.

Guster, D., O'Brien, A. Q., & Lebentritt, L. Can a Decentralized Structured Storage System such as Cassandra Provide an Effective Means of Speeding Up Web Access Times.

Hu, H., Wen, Y., Chua, T. S., & Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. IEEE Access, 2, 652-687.

IBM (2015a). The four v's of Big Data. Retrieved March 11, 2016 from http://www.ibmbigdatahub.com/infographic/four-vs-big-data

IBM (2015b). Why speed matters for big data and analytics. Retrieved Feburary 12, 2016 from http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ST&infotype=SA&appname=STGE_NI_EZ_USEN&htmlfid=NIJ12345USEN&attachment=NIJ12345USEN.PDF

Jacobs, A. (2009). The pathologies of big data. Communications of the ACM, 52(8), 36-44.

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., & Shahabi, C. (2014). Big data and its technical challenges. Communications of the ACM, 57(7), 86-94.

Jarr, Scott. (July, 2014). Part Three: Designing a Data Architecture to Support Both Fast and Big Data. Retrieved November 7, 2015, from https://voltdb.com/blog/part-three-designing-data-architecture-support-both-fast-and-big-data-0

Jewell, D., Barros, R. D., Diederichs, S., Duijvestijn, L. M., Hammersley, M., Hazra, A., ... & Portilla, I. (2014). Performance and capacity implications for big data. IBM Redbooks.

Klein, J., Gorton, I., Ernst, N., Donohoe, P., Pham, K., & Matser, C. (2014). Quality Attribute-

Guided Evaluation of NoSQL Databases: An Experience Report. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Logical Thoughts on Technology. (2013). Retrieved March 04, 2016 from http://www.lopakalogic.com/articles/hadoop-articles/hadoop-web-interface/

Lucas, R., Ang, J., Bergman, K., Borkar, S., Carlson, W., Carrington, L., ... & Geist, A. (2014). DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report: Top Ten Exascale Research Challenges. USDOE Office of Science (SC)(United States).

Morabito, V. (2015). Big data and analytics. Strategic and Organisational Impacts.

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of Big Data, 2(1), 1.

Reed, D. A., & Dongarra, J. (2015). Exascale computing and big data. Communications of the ACM, 58(7), 56-68.

Singh, D., & Reddy, C. K. (2014). A survey on platforms for big data analytics. Journal of Big Data, 2(1), 1.

Sqoop User Guide (v1.4.6). (n.d.). Retrieved January 04, 2016, from http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., & Rasin, A. (2010). MapReduce and parallel DBMSs: friends or foes?. Communications of the ACM, 53(1), 64-71.

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., ... & Murthy, R. (2009).

Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment, 2(2), 1626-1629.

Twitter API Overview. (n.d.). Retrieved February 12, 2016, from https://dev.twitter.com/overview/api

The Java EE 5 Tutorial. (n.d.). Retrieved March 12, 2016, from http://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html

Appendix

1. Auth.java

```
package com.scsu.auth;
import java.sql.SQLException;
import com.scsu.beans.User;
import com.scsu.facade.LoginFacadeImpl;
public class Auth {
	public static User ValidateUser(User user) throws SQLException {
		LoginFacadeImpl loginFacadeImpl = new LoginFacadeImpl();
		return loginFacadeImpl.validateLogin(user);
	}
}
```

2. DBAuth.java

```
package com.scsu.auth;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DBAuth {
	public static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
	public static final String HIVE_DRIVER = "org.apache.hive.jdbc.HiveDriver";
```

```
	public static final String MYSQL_Analysis_DB_URL =
"jdbc:mysql://10.59.7.90:3306/hadoopanalysis";
	public static final String HIVE_Analysis_DB_URL =
"jdbc:hive2://10.59.7.90:10000/default";
	public static final String MYSQL_Admin_DB_URL =
"jdbc:mysql://10.59.7.90:3306/administration";
	public static final String MYSQL_USER = "root";
	public static final String MYSQL_PASS = "root";
	public static final String HIVE_USER = "hiveuser";
	public static final String HIVE_PASS = "hivepassword";
	public Connection getMySQLAnalysisConnection() {
		Connection con = null;
		try {
			Class.forName(JDBC_DRIVER);
			con = DriverManager.getConnection(MYSQL_Analysis_DB_URL,
					MYSQL_USER, MYSQL_PASS);
		} catch (SQLException se) {
			se.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
		return con;
```

```
	}
	public Connection getMySQLAdminConnection() {
		Connection con = null;
		try {
			Class.forName(JDBC_DRIVER);
			con = DriverManager.getConnection(MYSQL_Admin_DB_URL,
MYSQL_USER,
					MYSQL_PASS);
		} catch (SQLException se) {
			se.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
		return con;
	}
	public Connection getHIVEAnalysisConnection() {
		Connection con = null;
		try {
			Class.forName(HIVE_DRIVER);
			con = DriverManager.getConnection(HIVE_Analysis_DB_URL,
HIVE_USER,
					HIVE_PASS);
```

```
		} catch (SQLException se) {
			se.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
		return con;
	}
}
```

3. User.java

```
package com.scsu.beans;
public class User {
	private boolean isValid;
	private String userName;
	private String password;
	public boolean isValid() {
		return isValid;
	}
	public void setValid(boolean isValid) {
		this.isValid = isValid;
	}
	public String getUserName() {
```

```
		return userName;
	}
	public void setUserName(String userName) {
		this.userName = userName;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
}
```

4. UserRole.java

```
package com.scsu.beans;
public class UserRole {
	private int roleId;
	private String roleName;
	public int getRoleId() {
		return roleId;
	}
	public void setRoleId(int roleId) {
```

```
		this.roleId = roleId;
	}
	public String getRoleName() {
		return roleName;
	}
	public void setRoleName(String roleName) {
		this.roleName = roleName;
	}
}
```

5. HIVEDAO.java

```
package com.scsu.dao;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
public interface HIVEDAO {
	public List<Map<String, Object>> executeSelect(String Query)
			throws SQLException;
	public List<Map<String, Object>> describeTable(String tableName)
			throws SQLException;
	public List<String> getDatabaseTables() throws SQLException;
}
```

6. HIVEDAOImpl.java

```
package com.scsu.dao;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import com.scsu.auth.DBAuth;
public class HIVEDAOImpl extends DBAuth implements HIVEDAO {
        public List<Map<String, Object>> executeSelect(String query)
                        throws SQLException {
                Connection connection = null;
                Statement statement = null;
                ResultSet resultSet = null;
                ResultSetMetaData resultSetMetaData = null;
                List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
```

```
        try {
                connection = getHIVEAnalysisConnection();
                statement = connection.createStatement();
                resultSet = statement.executeQuery(query);
                resultSetMetaData = resultSet.getMetaData();
                int columnCount = resultSetMetaData.getColumnCount();
                while (resultSet.next()) {
                        Map<String, Object> columns = new LinkedHashMap<String,
Object>();
                        for (int i = 1; i <= columnCount; i++) {
                                columns.put(resultSetMetaData.getColumnLabel(i),
                                                resultSet.getObject(i));
                        }
                        rows.add(columns);
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                resultSet.close();
                statement.close();
                connection.close();
```

```
		}
		return rows;
	}
	public List<Map<String, Object>> describeTable(String tableName)
			throws SQLException {
		Connection connection = null;
		Statement statement = null;
		ResultSet resultSet = null;
		ResultSetMetaData resultSetMetaData = null;
		List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
		try {
			connection = getHIVEAnalysisConnection();
			statement = connection.createStatement();
			resultSet = statement.executeQuery("select * from " + tableName
					+ " limit 1");
			resultSetMetaData = resultSet.getMetaData();
			int columnCount = resultSetMetaData.getColumnCount();
			while (resultSet.next()) {
				Map<String, Object> columns = new LinkedHashMap<String,
Object>();
				for (int i = 1; i <= columnCount; i++) {
					columns.put(resultSetMetaData.getColumnName(i),
```

```
resultSetMetaData.getColumnTypeName(i));
                        }
                        rows.add(columns);
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                resultSet.close();
                statement.close();
                connection.close();
        }
        return rows;
}
public List<String> getDatabaseTables() throws SQLException {
        Connection connection = null;
        ResultSet resultSet = null;
        DatabaseMetaData databaseMetaData = null;
        String[] dbTypes = { "TABLE" };
        ArrayList<String> hiveTables = new ArrayList<String>();
        try {
```

```
                connection = getHIVEAnalysisConnection();
                databaseMetaData = connection.getMetaData();
                resultSet = databaseMetaData.getTables(null, null, "%", dbTypes);
                while (resultSet.next()) {
                        hiveTables.add(resultSet.getString("TABLE_NAME"));
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                resultSet.close();
                connection.close();
        }
        return hiveTables;
    }
}
```

7. MySQLDAO.java

```
package com.scsu.dao;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
```

```
public interface MySQLDAO {
	public List<Map<String, Object>> executeSelect(String Query)
			throws SQLException;
	public List<Map<String, Object>> describeTable(String tableName)
			throws SQLException;
	public List<String> getDatabaseMySQLTables() throws SQLException;
}
```

8. MySQLDAOImpl.java

```
package com.scsu.dao;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import com.scsu.auth.DBAuth;
public class MySQLDAOImpl extends DBAuth implements MySQLDAO {
```

```
public List<Map<String, Object>> executeSelect(String query)
        throws SQLException {
    Connection connection = null;
    Statement statement = null;
    ResultSet resultSet = null;
    ResultSetMetaData resultSetMetaData = null;
    List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
    try {
        connection = getMySQLAnalysisConnection();
        statement = connection.createStatement();
        resultSet = statement.executeQuery(query);
        resultSetMetaData = resultSet.getMetaData();
        int columnCount = resultSetMetaData.getColumnCount();
        while (resultSet.next()) {
            Map<String, Object> columns = new LinkedHashMap<String,
Object>();
            for (int i = 1; i <= columnCount; i++) {
                columns.put(resultSetMetaData.getColumnLabel(i),
                        resultSet.getObject(i));
            }
            rows.add(columns);
        }
```

```
		} catch (SQLException se) {
			se.printStackTrace();
			throw new SQLException();
		} finally {
			resultSet.close();
			statement.close();
			connection.close();
		}
		return rows;
	}
	public List<Map<String, Object>> describeTable(String tableName)
			throws SQLException {
		Connection connection = null;
		Statement statement = null;
		ResultSet resultSet = null;
		ResultSetMetaData resultSetMetaData = null;
		List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
		try {
			connection = getMySQLAnalysisConnection();
			statement = connection.createStatement();
			resultSet = statement.executeQuery("select * from " + tableName
					+ " limit 1");
```

```
                resultSetMetaData = resultSet.getMetaData();
                int columnCount = resultSetMetaData.getColumnCount();
                while (resultSet.next()) {
                        Map<String, Object> columns = new LinkedHashMap<String,
Object>();
                        for (int i = 1; i <= columnCount; i++) {
                                columns.put(resultSetMetaData.getColumnName(i),

        resultSetMetaData.getColumnTypeName(i));
                        }
                        rows.add(columns);
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                resultSet.close();
                statement.close();
                connection.close();
        }
        return rows;
    }
```

```
public List<String> getDatabaseMySQLTables() throws SQLException {
        Connection connection = null;
        ResultSet resultSet = null;
        DatabaseMetaData databaseMetaData = null;
        String[] dbTypes = { "TABLE" };
        ArrayList<String> mysqlTables = new ArrayList<String>();
        try {
                connection = getMySQLAnalysisConnection();
                databaseMetaData = connection.getMetaData();
                resultSet = databaseMetaData.getTables(null, null, "%", dbTypes);
                while (resultSet.next()) {
                        mysqlTables.add(resultSet.getString("TABLE_NAME"));
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                resultSet.close();
                connection.close();
        }
        return mysqlTables;
}
```

```
}
```

9. LoginDAO.java

```
package com.scsu.dao;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import com.scsu.beans.User;
import com.scsu.beans.UserRole;
public interface LoginDAO {
	public User validateLogin(User user) throws SQLException;
	public int getUserId(User user) throws SQLException;
	public List<UserRole> getUserRoles(int userId) throws SQLException;
	public List<Map<String, Object>> getQueryComparisionData()
			throws SQLException;
	public void insertQueryData(String query, long mysqlTime, long hiveTime)
			throws SQLException;
	public String getChartData() throws SQLException;
}
```

10. LoginDAOImpl.java

```
package com.scsu.dao;
```

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import com.scsu.auth.DBAuth;
import com.scsu.beans.User;
import com.scsu.beans.UserRole;
public class LoginDAOImpl extends DBAuth implements LoginDAO {
	public User validateLogin(User user) throws SQLException {
		Connection connection = null;
		Statement statement = null;
		ResultSet resultSet = null;
		boolean isValid = false;
		try {
			connection = getMySQLAdminConnection();
			statement = connection.createStatement();
			String sql = "SELECT ACTIVE FROM users where username=" + "'"
```

```
                    + user.getUserName() + "'" + " and password=" + "'"
                    + user.getPassword() + "'" + ";";
            resultSet = statement.executeQuery(sql);
            while (resultSet.next()) {
                if (resultSet.getBoolean("ACTIVE")) {
                    isValid = true;
                }
            }
            user.setValid(isValid);
        } catch (SQLException se) {
            se.printStackTrace();
            throw new SQLException();
        } finally {
            resultSet.close();
            statement.close();
            connection.close();
        }
        return user;
    }
    public int getUserId(User user) throws SQLException {
        Connection connection = null;
        Statement statement = null;
```

```
        ResultSet resultSet = null;
        int userId = 0;
        try {
                connection = getMySQLAdminConnection();
                statement = connection.createStatement();
                String sql = "SELECT USER_ID FROM users where username=" + "'"
                                + user.getUserName() + "'" + " and password=" + "'"
                                + user.getPassword() + "'" + ";";
                resultSet = statement.executeQuery(sql);
                while (resultSet.next()) {
                        userId = resultSet.getInt(1);
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                resultSet.close();
                statement.close();
                connection.close();
        }
        return userId;
}
```

```
public List<UserRole> getUserRoles(int userId) throws SQLException {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        List<UserRole> userRoles = new ArrayList<UserRole>();
        try {
                connection = getMySQLAdminConnection();
                statement = connection.createStatement();
                String sql;
                sql = "SELECT USER_ROLE_ID, AUTHORITY FROM user_roles
where user_id="
                                + userId + ";";
                resultSet = statement.executeQuery(sql);
                while (resultSet.next()) {
                        UserRole userRole = new UserRole();
                        userRole.setRoleId(resultSet.getInt(1));
                        userRole.setRoleName(resultSet.getString(2));
                        userRoles.add(userRole);
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
```

```
		} finally {
			resultSet.close();
			statement.close();
			connection.close();
		}
		return userRoles;
	}
	public List<Map<String, Object>> getQueryComparisionData()
			throws SQLException {
		Connection connection = null;
		Statement statement = null;
		ResultSet resultSet = null;
		ResultSetMetaData resultSetMetaData = null;
		List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
		try {
			connection = getMySQLAdminConnection();
			statement = connection.createStatement();
			String sql = "SELECT Query,avg(MySQLTime) as 'Avg MySQL
Time',avg(HIVETime) as 'Avg HIVE Time' FROM queries group by Query;";
			resultSet = statement.executeQuery(sql);

			resultSetMetaData = resultSet.getMetaData();
```

```
                int columnCount = resultSetMetaData.getColumnCount();
                while (resultSet.next()) {
                        Map<String, Object> columns = new LinkedHashMap<String,
Object>();
                        for (int i = 1; i <= columnCount; i++) {
                                columns.put(resultSetMetaData.getColumnLabel(i),
                                                resultSet.getObject(i));
                        }
                        rows.add(columns);
                }
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                resultSet.close();
                statement.close();
                connection.close();
        }
        return rows;
    }
    public void insertQueryData(String query, long mysqlTime, long hiveTime)
                throws SQLException {
```

```
        Connection connection = null;
        Statement statement = null;
        try {
                connection = getMySQLAdminConnection();
                statement = connection.createStatement();
                String sql = "insert into queries (Query, MySQLTime, HIVETime) "
                                + "values('" + query + "'," + mysqlTime + "," + hiveTime
                                + ");";
                statement.executeUpdate(sql);
        } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
        } finally {
                statement.close();
                connection.close();
        }
}
public String getChartData() throws SQLException {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        String data = "";
```

```
            try {
                connection = getMySQLAdminConnection();
                statement = connection.createStatement();
                String sql = "SELECT Query,avg(MySQLTime) as 'Avg MySQL
Time',avg(HIVETime) as 'Avg HIVE Time' FROM queries group by Query;";
                resultSet = statement.executeQuery(sql);
                while (resultSet.next()) {
                    data += "['" + resultSet.getString("Query") + "',";
                    data += resultSet.getInt("Avg MySQL Time") + ",";
                    data += resultSet.getInt("Avg HIVE Time") + "],";
                }
            } catch (SQLException se) {
                se.printStackTrace();
                throw new SQLException();
            } finally {
                resultSet.close();
                statement.close();
                connection.close();
            }
            return data;
        }
}
```

11. CommonFacade.java

```
package com.scsu.facade;
public interface CommonFacade {
	public boolean parseQuery(String query);
}
```

12. CommonFacadeImpl.java

```
package com.scsu.facade;
public class CommonFacadeImpl implements CommonFacade {
	public boolean parseQuery(String query) {
		boolean isValid = false;
		if (query != null && query.contains("select")) {
			isValid = true;
		}
		return isValid;
	}
}
```

13. HIVEFacade.java

```
package com.scsu.facade;
import java.sql.SQLException;
```

```
import java.util.List;
import java.util.Map;
public interface HIVEFacade {
	public List<Map<String, Object>> executeSelect(String query)
			throws SQLException;
	public List<Map<String, Object>> describeTable(String tableName)
			throws SQLException;
	public List<String> getDatabaseTables() throws SQLException;
}
```

14. HIVEFacadeImpl.java

```
package com.scsu.facade;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import com.scsu.dao.HIVEDAOImpl;
public class HIVEFacadeImpl extends CommonFacadeImpl implements HIVEFacade {
	public List<Map<String, Object>> executeSelect(String query) throws SQLException {
		HIVEDAOImpl hivedaoImpl = new HIVEDAOImpl();
		return hivedaoImpl.executeSelect(query);
	}
```

```
	public List<Map<String, Object>> describeTable(String tableName) throws
SQLException {
		HIVEDAOImpl hivedaoImpl = new HIVEDAOImpl();
		return hivedaoImpl.describeTable(tableName);
	}
	public List<String> getDatabaseTables() throws SQLException{
		HIVEDAOImpl hivedaoImpl = new HIVEDAOImpl();
		return hivedaoImpl.getDatabaseTables();
	}
}
```

15. HIVEThread.java

```
package com.scsu.facade;
import java.sql.SQLException;
public class HiveThread extends Thread {
	private String query;
	private boolean error;
	public void setError(boolean error) {
		this.error = error;
	}
	public boolean getError() {
		return error;
```

```
	}
	public HiveThread(String query) {
		this.query = query;
	}
	public void run() {
		HIVEFacadeImpl hiveFacadeImpl = new HIVEFacadeImpl();
		try {
			hiveFacadeImpl.executeSelect(query);
		} catch (SQLException e) {
			setError(true);
			System.out.println("SQL Exception in HIVE Thread");
		}
	}
}
```

16. LoginFacade.java

```
package com.scsu.facade;

import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import com.scsu.beans.User;
```

```
import com.scsu.beans.UserRole;
public interface LoginFacade {
	public User validateLogin(User user) throws SQLException;
	public int getUserId(User user) throws SQLException;
	public List<UserRole> getUserRoles(int userId) throws SQLException;
	public List<Map<String, Object>> getQueryComparisionData()
			throws SQLException;
	public void insertQueryData(String query, long mysqlTime, long hiveTime)
			throws SQLException;
	public String getChartData() throws SQLException;
}
```

17. LoginFacadeImpl.java

```
package com.scsu.facade;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import com.scsu.beans.User;
import com.scsu.beans.UserRole;
import com.scsu.dao.LoginDAOImpl;
public class LoginFacadeImpl implements LoginFacade {
	public User validateLogin(User user) throws SQLException {
		LoginDAOImpl loginDAOImpl = new LoginDAOImpl();
```

```
        return loginDAOImpl.validateLogin(user);
}
public List<UserRole> getUserRoles(int userId) throws SQLException {
        LoginDAOImpl loginDAOImpl = new LoginDAOImpl();
        return loginDAOImpl.getUserRoles(userId);
}
public int getUserId(User user) throws SQLException {
        LoginDAOImpl loginDAOImpl = new LoginDAOImpl();
        return loginDAOImpl.getUserId(user);
}
public List<Map<String, Object>> getQueryComparisionData() throws SQLException {
        LoginDAOImpl loginDAOImpl = new LoginDAOImpl();
        return loginDAOImpl.getQueryComparisionData();
}
public void insertQueryData(String query, long mysqlTime, long hiveTime)
                throws SQLException {
        LoginDAOImpl loginDAOImpl = new LoginDAOImpl();
        loginDAOImpl.insertQueryData(query, mysqlTime, hiveTime);
}
public String getChartData() throws SQLException{
        LoginDAOImpl loginDAOImpl = new LoginDAOImpl();
        return loginDAOImpl.getChartData();
```

```
	}
}
```

18. MySQLFacade.java

```
package com.scsu.facade;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
public interface MySQLFacade {
	public List<Map<String, Object>> executeSelect(String query) throws SQLException;
	public List<Map<String, Object>>  describeTable(String tableName) throws
SQLException;
	public List<String> getDatabaseTables() throws SQLException;
}
```

19. MySQLFacadeImpl.java

```
package com.scsu.facade;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import com.scsu.dao.MySQLDAOImpl;
public class MySQLFacadeImpl extends CommonFacadeImpl implements MySQLFacade {
```

```
	public List<Map<String, Object>> executeSelect(String query) throws SQLException {
		MySQLDAOImpl mySQLDAOImpl = new MySQLDAOImpl();
		return mySQLDAOImpl.executeSelect(query);
	}
	public List<Map<String, Object>>  describeTable(String tableName) throws
SQLException {
		MySQLDAOImpl mySQLDAOImpl = new MySQLDAOImpl();
		return mySQLDAOImpl.describeTable(tableName);
	}
	public List<String> getDatabaseTables() throws SQLException {
		MySQLDAOImpl mySQLDAOImpl = new MySQLDAOImpl();
		return mySQLDAOImpl.getDatabaseMySQLTables();
	}
}
```

20. MySQLThread.java

```
package com.scsu.facade;
import java.sql.SQLException;
public class MySQLThread extends Thread {
	private String query;
	private boolean error;
	public void setError(boolean error) {
```

```
            this.error = error;
        }
        public boolean getError() {
            return error;
        }
        public MySQLThread(String query) {
            this.query = query;
        }
        public void run() {
            MySQLFacadeImpl mysqlFacadeImpl = new MySQLFacadeImpl();
            try {
                mysqlFacadeImpl.executeSelect(query);
            } catch (SQLException e) {
                setError(true);
                System.out.println("SQL Exception in MySQL Thread");
            }
        }
}
```

21. ChartsServlet.java

```
package com.scsu.servlets;
import java.io.IOException;
```

```
import java.sql.SQLException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.scsu.facade.LoginFacadeImpl;
public class ChartsServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	protected void doGet(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		doPost(request, response);
	}
	protected void doPost(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		LoginFacadeImpl loginFacadeImpl = new LoginFacadeImpl();
		String data = "[['Query', 'Avg MySQL Time', 'Avg HIVE Time'],";
		try {
			data+=loginFacadeImpl.getChartData()+"]";
		} catch (SQLException e) {
			e.printStackTrace();
		}
```

```
                request.setAttribute("data", data);
                RequestDispatcher requestDispatcher = request
                                .getRequestDispatcher("/charts.jsp");
                requestDispatcher.forward(request, response);
        }
}
```

22. DescribeTable.java

```
package com.scsu.servlets;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.scsu.facade.HIVEFacadeImpl;
import com.scsu.facade.MySQLFacadeImpl;
public class DescribeTable extends HttpServlet {
        private static final long serialVersionUID = 1L;
```

```
public DescribeTable() {
        super();
}
protected void doGet(HttpServletRequest request,
                HttpServletResponse response) throws ServletException, IOException {
        doPost(request, response);
}
protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws ServletException, IOException {
        String tableName = request.getParameter("tableName");
        String databaseName = request.getParameter("databaseName");
        HIVEFacadeImpl hiveFacadeImpl = new HIVEFacadeImpl();
        MySQLFacadeImpl mysqlFacadeImpl = new MySQLFacadeImpl();
        List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
        try {
                if (databaseName.equalsIgnoreCase("mysql"))
                        rows = mysqlFacadeImpl.describeTable(tableName);
                else if (databaseName.equalsIgnoreCase("hive"))
                        rows = hiveFacadeImpl.describeTable(tableName);
        } catch (SQLException e) {
                e.printStackTrace();
        }
```

```
                request.setAttribute("describeTableResult", rows);
                request.getRequestDispatcher("/describetable.jsp").include(request,
                                response);
        }
}
```

23. HIVEQueryProcessor.java

```
package com.scsu.servlets;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.scsu.facade.HIVEFacadeImpl;
public class HIVEQueryProcessor extends HttpServlet {
        private static final long serialVersionUID = 1L;
        public HIVEQueryProcessor() {
                super();
        }
```

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    doPost(request, response);
}
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    String query = request.getParameter("hiveQuery");
    HIVEFacadeImpl hiveFacadeImpl = new HIVEFacadeImpl();
    List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
    boolean validQuery = hiveFacadeImpl.parseQuery(query);
    if (validQuery) {
        try {
            rows = hiveFacadeImpl.executeSelect(query);
        } catch (Exception e) {
            request.setAttribute("error", "Invalid HIVE Query");
            request.getRequestDispatcher("/invalidquery.jsp").include(
                    request, response);
        }
        request.setAttribute("hiveresult", rows);
        request.getRequestDispatcher("/hiveresult.jsp").include(request,
                response);
    } else {
```

```
                        request.setAttribute("error", "Invalid HIVE Query");
                        request.getRequestDispatcher("/invalidquery.jsp").include(request,
                                        response);
                }
        }
}
```

24. MySQLQueryProcessor.java

```
package com.scsu.servlets;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.scsu.facade.MySQLFacadeImpl;
public class MySQLQueryProcessor extends HttpServlet {
        private static final long serialVersionUID = 1L;
        public MySQLQueryProcessor() {
                super();
```

```
}
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    doPost(request, response);
}
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    String query = request.getParameter("mysqlQuery");
    MySQLFacadeImpl mySQLFacadeImpl = new MySQLFacadeImpl();
    List<Map<String, Object>> rows = new ArrayList<Map<String, Object>>();
    boolean validQuery = mySQLFacadeImpl.parseQuery(query);
    if (validQuery) {
        try {
            rows = mySQLFacadeImpl.executeSelect(query);
        } catch (Exception e) {
            request.setAttribute("error", "Invalid MySQL Query");
            request.getRequestDispatcher("/invalidquery.jsp").include(
                    request, response);
        }
        request.setAttribute("mysqlresult", rows);
        request.getRequestDispatcher("/mysqlresult.jsp").include(request,
                response);
```

```
                } else {
                        request.setAttribute("error", "Invalid MySQL Query");
                        request.getRequestDispatcher("/invalidquery.jsp").include(request,
                                        response);
                }
        }
}
```

25. CompareQueryProcessor.java

```
package com.scsu.servlets;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.scsu.facade.CommonFacadeImpl;
import com.scsu.facade.HiveThread;
import com.scsu.facade.LoginFacadeImpl;
```

```
import com.scsu.facade.MySQLThread;
public class CompareQueryProcessor extends HttpServlet {
	private static final long serialVersionUID = 1L;
	public CompareQueryProcessor() {
		super();
	}
	protected void doGet(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		doPost(request, response);
	}
	protected void doPost(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		String query = request.getParameter("compareQuery");
		CommonFacadeImpl commonFacadeImpl = new CommonFacadeImpl();
		boolean validQuery = commonFacadeImpl.parseQuery(query);
		if (validQuery) {
			long mysqltime = 0, hivetime = 0;
			LoginFacadeImpl loginFacadeImpl = new LoginFacadeImpl();
			List<Map<String, Object>> rows = new ArrayList<Map<String,
Object>>();
			MySQLThread mt = new MySQLThread(query);
			HiveThread ht = new HiveThread(query);
```

```
long start = System.nanoTime();
long mtEnd = 0;
long htEnd = 0;
mt.start();
ht.start();
while (mt.isAlive() || ht.isAlive()) {
        if (mt.isAlive())
                mtEnd = System.nanoTime();
        if (ht.isAlive())
                htEnd = System.nanoTime();
}
mysqltime = (mtEnd - start)/1000000;
hivetime = (htEnd - start)/1000000;
if (!(mt.getError() || ht.getError())) {
        try {
                loginFacadeImpl.insertQueryData(query, mysqltime,
hivetime);
                rows = loginFacadeImpl.getQueryComparisionData();
        } catch (SQLException se) {
                System.out.println("SQL Exception");
        }
        request.setAttribute("qData", rows);
```

```
                        request.setAttribute("mysqltime", mysqltime);
                        request.setAttribute("hivetime", hivetime);
                        request.getRequestDispatcher("/comparequery.jsp").include(
                                request, response);

                } else {
                        request.setAttribute("error", "Invalid Query");
                        request.getRequestDispatcher("/invalidquery.jsp").include(
                                request, response);
                }
        } else {
                request.setAttribute("error", "Invalid Query");
                request.getRequestDispatcher("/invalidquery.jsp").include(request,
                        response);
        }
    }
}
```

26. LoginServlet.java

```
package com.scsu.servlets;
import java.io.IOException;
import java.sql.SQLException;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.scsu.auth.Auth;
import com.scsu.beans.User;
import com.scsu.beans.UserRole;
import com.scsu.facade.HIVEFacadeImpl;
import com.scsu.facade.LoginFacadeImpl;
import com.scsu.facade.MySQLFacadeImpl;
public class LoginServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	protected void doGet(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		doPost(request, response);
	}
```

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    String userName = request.getParameter("username");
    String password = request.getParameter("password");
    User user = new User();
    user.setUserName(userName.trim());
    user.setPassword(password.trim());
    boolean adminAccess = false;
    List<UserRole> uRoles = new ArrayList<UserRole>();
    LoginFacadeImpl loginFacadeImpl = new LoginFacadeImpl();
    MySQLFacadeImpl mySQLFacadeImpl = new MySQLFacadeImpl();
    HIVEFacadeImpl hiveFacadeImpl = new HIVEFacadeImpl();
    List<Map<String, Object>> comparisionData = new ArrayList<Map<String,
Object>>();
    List<String> mysqlTables = new ArrayList<String>();
    List<String> hiveTables = new ArrayList<String>();
    if (userName != null && userName.trim().length() > 0
            && password != null && password.trim().length() > 0) {
        try {
            user = Auth.ValidateUser(user);
        } catch (SQLException e) {
            e.printStackTrace();
```

```
                }
                if (user.isValid()) {
                        try {
                                uRoles = loginFacadeImpl.getUserRoles(loginFacadeImpl
                                                .getUserId(user));
                                for (UserRole ur : uRoles) {
                                        if
(ur.getRoleName().equalsIgnoreCase("ROLE_ADMIN")) {
                                                adminAccess = true;
                                                break;
                                        }
                                }
                                mysqlTables = mySQLFacadeImpl.getDatabaseTables();
                                hiveTables = hiveFacadeImpl.getDatabaseTables();
                                comparisionData =
loginFacadeImpl.getQueryComparisionData();
                        } catch (SQLException se) {
                                se.printStackTrace();
                        }
                        if (request.getParameter("remember") != null) {
                                String remember = request.getParameter("remember");
```

```
                    Cookie cUserName = new Cookie("cookuser",
userName.trim());
                    Cookie cPassword = new Cookie("cookpass",
userName.trim());
                    Cookie cRemember = new Cookie("cookrem",
remember.trim());
                    cUserName.setMaxAge(60 * 60 * 24 * 15);// 15 days
                    cPassword.setMaxAge(60 * 60 * 24 * 15);
                    cRemember.setMaxAge(60 * 60 * 24 * 15);
                    response.addCookie(cUserName);
                    response.addCookie(cPassword);
                    response.addCookie(cRemember);
                }
                HttpSession httpSession = request.getSession();
                httpSession.setAttribute("sessuser", userName.trim());
                request.setAttribute("adminAccess", adminAccess);
                request.setAttribute("mysqlTables", mysqlTables);
                request.setAttribute("hiveTables", hiveTables);
                request.setAttribute("qData", comparisionData);
                RequestDispatcher requestDispatcher = request
                        .getRequestDispatcher("/home.jsp");
                requestDispatcher.forward(request, response);
```

```
                } else {
                        System.out.println("Authentication failure.");
                        request.setAttribute("error", "Authentication failure.");
                        RequestDispatcher requestDispatcher = request
                                        .getRequestDispatcher("/login.jsp");
                        requestDispatcher.forward(request, response);
                }
        } else {
                System.out.println("Username and Password are required fields.");
                request.setAttribute("error",
                                "Username and Password are required fields.");
                RequestDispatcher requestDispatcher = request
                                .getRequestDispatcher("/login.jsp");
                requestDispatcher.forward(request, response);
        }
    }
}
```

27. LogoutServlet.java

```
package com.scsu.servlets;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LogoutServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	protected void doGet(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		doPost(request, response);
	}
	protected void doPost(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {
		Cookie cUserName = new Cookie("cookuser", null);
		Cookie cPassword = new Cookie("cookpass", null);
		Cookie cRemember = new Cookie("cookrem", null);
		cUserName.setMaxAge(0);
		cPassword.setMaxAge(0);
		cRemember.setMaxAge(0);
		response.addCookie(cUserName);
		response.addCookie(cPassword);
```

```
                response.addCookie(cRemember);
                HttpSession httpSession = request.getSession();
                httpSession.invalidate();
                request.setAttribute("msg", "You have successfully logged out.");
                RequestDispatcher requestDispatcher = request
                                .getRequestDispatcher("/login.jsp");
                requestDispatcher.forward(request, response);
        }
}
```

28. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>Hadoop-Analysis</display-name>
  <welcome-file-list>
    <welcome-file>login.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <description>Used to login a user using credentials</description>
```

```
  <display-name>LoginServlet</display-name>
  <servlet-name>LoginServlet</servlet-name>
  <servlet-class>com.scsu.servlets.LoginServlet</servlet-class>
</servlet>
<servlet>
  <description>Used to logout a logged in user</description>
  <display-name>LogoutServlet</display-name>
  <servlet-name>LogoutServlet</servlet-name>
  <servlet-class>com.scsu.servlets.LogoutServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>LoginServlet</servlet-name>
  <url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>LogoutServlet</servlet-name>
  <url-pattern>/LogoutServlet</url-pattern>
</servlet-mapping>
<servlet>
  <description></description>
  <display-name>MySQLQueryProcessor</display-name>
  <servlet-name>MySQLQueryProcessor</servlet-name>
```

```
  <servlet-class>com.scsu.servlets.MySQLQueryProcessor</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MySQLQueryProcessor</servlet-name>
  <url-pattern>/MySQLQueryProcessor</url-pattern>
</servlet-mapping>
<servlet>
  <description></description>
  <display-name>HIVEQueryProcessor</display-name>
  <servlet-name>HIVEQueryProcessor</servlet-name>
  <servlet-class>com.scsu.servlets.HIVEQueryProcessor</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HIVEQueryProcessor</servlet-name>
  <url-pattern>/HIVEQueryProcessor</url-pattern>
</servlet-mapping>
<servlet>
  <description></description>
  <display-name>ChartQueryProcessor</display-name>
  <servlet-name>ChartQueryProcessor</servlet-name>
  <servlet-class>com.scsu.servlets.ChartQueryProcessor</servlet-class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>ChartQueryProcessor</servlet-name>
  <url-pattern>/ChartQueryProcessor</url-pattern>
</servlet-mapping>
<servlet>
  <description></description>
  <display-name>CompareQueryProcessor</display-name>
  <servlet-name>CompareQueryProcessor</servlet-name>
  <servlet-class>com.scsu.servlets.CompareQueryProcessor</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>CompareQueryProcessor</servlet-name>
  <url-pattern>/CompareQueryProcessor</url-pattern>
</servlet-mapping>
<servlet>
  <description></description>
  <display-name>DescribeTable</display-name>
  <servlet-name>DescribeTable</servlet-name>
  <servlet-class>com.scsu.servlets.DescribeTable</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>DescribeTable</servlet-name>
```

```
    <url-pattern>/DescribeTable</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>ChartsServlet</display-name>
    <servlet-name>ChartsServlet</servlet-name>
    <servlet-class>com.scsu.servlets.ChartsServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ChartsServlet</servlet-name>
    <url-pattern>/ChartsServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

29. charts.jsp

```
<html>
<head>
<script type="text/javascript"
        src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
        google.charts.load('current', {
                'packages' : [ 'corechart' ]
```

```
	});
	google.charts.setOnLoadCallback(drawChart);
	function drawChart() {
		var data = google.visualization.arrayToDataTable(${data});
		var options = {
			hAxis : {
				title : 'Queries'
			},
			vAxis : {
				title : 'Time Taken in Milli Seconds'
			},
			colors : [ '#a52714', '#097138' ]
		};
		var chart = new google.visualization.LineChart(document
				.getElementById('linechart'));
		chart.draw(data, options);
	}
</script>
</head>
<body>
	<div id="linechart" style="width: 900px; height: 500px"></div>
</body>
```

```
</html>
```

30. describetable.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<table id="describetable">
	<thead>
		<tr>
			<c:forEach items="${describeTableResult[0]}" var="column">
				<td><c:out value="${column.key}" /></td>
			</c:forEach>
		</tr>
	</thead>
	<tbody>
		<c:forEach items="${describeTableResult}" var="columns">
			<tr>
				<c:forEach items="${columns}" var="column">
					<td><c:out value="${column.value}" /></td>
				</c:forEach>
			</tr>
		</c:forEach>
	</tbody>
</table>
```

31. comparequery.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<label>Time taken by MySQL:</label>${mysqltime}
<br />
<label>Time taken by HIVE :</label>${hivetime}
<table>
	<thead>
		<tr>
			<c:forEach items="${qData[0]}" var="column">
				<td><c:out value="${column.key}" /></td>
			</c:forEach>
		</tr>
	</thead>
	<tbody>
		<c:forEach items="${qData}" var="columns">
			<tr>
				<c:forEach items="${columns}" var="column">
					<td><c:out value="${column.value}" /></td>
				</c:forEach>
			</tr>
		</c:forEach>
```

```
        </tbody>
</table>
```

32. header.jsp

```
<img alt="" id="header-logo" src="header-logo.png">
```

33. invalidquery.jsp

```
<%=request.getAttribute("error") != null ? request
                                        .getAttribute("error") : ""%>
```

34. mysqlresult.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<table>
        <thead>
                <tr>
                        <c:forEach items="${hiveresult[0]}" var="column">
                                <td><c:out value="${column.key}" /></td>
                        </c:forEach>
                </tr>
        </thead>
        <tbody>
                <c:forEach items="${hiveresult}" var="columns">
```

```
                <tr>
                    <c:forEach items="${columns}" var="column">
                        <td><c:out value="${column.value}" /></td>
                    </c:forEach>
                </tr>
            </c:forEach>
        </tbody>
</table>
```

35. mysqlresult.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<table id="mysqltable">
    <thead>
        <tr>
            <c:forEach items="${mysqlresult[0]}" var="column">
                <td><c:out value="${column.key}" /></td>
            </c:forEach>
        </tr>
    </thead>
    <tbody>
        <c:forEach items="${mysqlresult}" var="columns">
            <tr>
```

```
                        <c:forEach items="${columns}" var="column">
                            <td><c:out value="${column.value}" /></td>
                        </c:forEach>
                    </tr>
                </c:forEach>
            </tbody>
</table>
```

36. home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Home Page</title>
<link rel="stylesheet" type="text/css" href="style.css">
<script src="https://code.jquery.com/jquery-1.11.3.js"></script>
<script src="http://code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
<script type="text/javascript">
    $(document)
```

```
.ready(
        function(event) {
            $("#mysqlParse")
                .click(
                    function() {
                        $
.get(
    "MySQLQueryProcessor",
    {
        mysqlQuery : $(
            "#mysqlQuery")
            .val(),
    },
    function(data,
```

```
                status) {

            document

                    .getElementById("mysqlResult").innerHTML = data;

    });
                                                        });
                                    $("#hiveParse")
                                                .click(
                                                        function() {
                                                            $

.get(

        "HIVEQueryProcessor",

        {

            hiveQuery : $(
```

```
                "#hiveQuery")

                .val(),

},

function(data,

        status) {

    document

                .getElementById("hiveResult").innerHTML = data;

});
                                                    });
                    $("#compareParse")
                                .click(
                                                    function() {
                                                        $
```

```
.get(
        "CompareQueryProcessor",
        {
                compareQuery : $(
                                "#compareQuery")
                                .val(),
        },
        function(data,
                        status) {
                if (data != "Invalid Query")
                        $(
```

```
                                        "#resultPriorData")

                                        .hide();

                        document

                                .getElementById("compareResult").innerHTML = data;

                });
                                                                        });
                                        $("#mysqlexportxls")
                                                .click(
                                                                        function(e) {
                                                                                if
(String($('#mysqlResult').html())

                .includes("table"))

                window

                .open('data:application/vnd.ms-excel,'
```

```
                + encodeURIComponent($(

                        '#mysqlResult')

                        .html()));

        e.preventDefault();
                                                        });
                                $("#hiveexportxls")
                                        .click(
                                                function(e) {
                                                        if
(String($('#hiveResult').html())

        .includes("table"))

        window

        .open('data:application/vnd.ms-excel,'

                + encodeURIComponent($(
```

```
                                '#hiveResult')

                                .html()));

e.preventDefault();
                                                                    });
                                        $("#compareexportxls")
                                                .click(
                                                                function(e) {
                                                                        if (String(

$('#compareResult').html())

.includes("table"))

window

.open('data:application/vnd.ms-excel,'

                + encodeURIComponent($(
```

```
                                    '#compareResult')

                                    .html()));

e.preventDefault();
                                                                    });
                                    });

$(function() {
        $('.tab-section').hide();
        $('#tabs a').bind('click', function(e) {
                $('#tabs a.current').removeClass('current');
                $('.tab-section:visible').hide();
                $(this.hash).show();
                $(this).addClass('current');
                e.preventDefault();
        }).filter(':first').click();
});
$(function() {
        $('.inner-tab-section').hide();
        $('#subtabs a').bind('click', function(e) {
```

```
                $('#subtabs a.current').removeClass('current');
                $('.inner-tab-section:visible').hide();
                $(this.hash).show();
                $(this).addClass('current');
                e.preventDefault();
        }).filter(':first').click();
});
function describeMySQLTable(table) {
        $.get("DescribeTable", {
                tableName : table,
                databaseName : "mysql",
        }, function(data, status) {
                document.getElementById("mysqlTableDesc").innerHTML = data;
        });
}
function describeHIVETable(table) {
        $.get("DescribeTable", {
                tableName : table,
                databaseName : "hive",
        }, function(data, status) {
                document.getElementById("hiveTableDesc").innerHTML = data;
        });
```

```
        }
</script>
</head>
<body>
        <jsp:include page="header.jsp"></jsp:include>
        <p>
                <label id="welcome">Welcome, <%=session.getAttribute("sessuser")%></label>
                <a id="logout"
href="<%=request.getContextPath()%>/LogoutServlet">Logout</a>
        </p>
        <h2>Interact with MySQL and HIVE databases</h2>
        <ul id="tabs">
                <li><a href="#mysql">MySQL</a></li>
                <li><a href="#hive">HIVE</a></li>
                <li><a href="#compare">Performance</a></li>
                <li><a href="#guide">User Guide</a></li>
        </ul>
        <div id="mysql" class="tab-section">
                <h2>MySQL Query Processor</h2>
                <input type="text" id="mysqlQuery">
                <c:if test="${adminAccess}">
                        <img src="excel.png" alt="" id="mysqlexportxls">
```

```
            </c:if>
            <input type="button" id="mysqlParse" value="Run Query" /> <br />
            <div id="mysqlResult"></div>
        </div>
        <div id="hive" class="tab-section">
            <h2>HIVE Query Processor</h2>
            <input type="text" id="hiveQuery">
            <c:if test="${adminAccess}">
                <img src="excel.png" alt="" id="hiveexportxls">
            </c:if>
            <input type="button" id="hiveParse" value="Run Query" /> <br />
            <div id="hiveResult"></div>
        </div>
        <div id="compare" class="tab-section">
            <h2>Time Comparision with MySQL and HIVE</h2>
            <input type="text" id="compareQuery">
            <c:if test="${adminAccess}">
                <img src="excel.png" alt="" id="compareexportxls">
                <a id="charts"
                    href="<%=request.getContextPath()%>/ChartsServlet"
target="_blank">Charts</a>
            </c:if>
```

```
<input type="button" id="compareParse" value="Run Query" /> <br />
<div id="compareResult">
    <table id="resultPriorData">
        <thead>
            <tr>
                <c:forEach items="${qData[0]}" var="column">
                    <td><c:out value="${column.key}" /></td>
                </c:forEach>
            </tr>
        </thead>
        <tbody>
            <c:forEach items="${qData}" var="columns">
                <tr>
                    <c:forEach items="${columns}"
var="column">
                        <td><c:out
value="${column.value}" /></td>
                    </c:forEach>
                </tr>
            </c:forEach>
        </tbody>
    </table>
```

```
            </div>
        </div>
        <div id="guide" class="tab-section">
            <h2>User Guide</h2>
            <ul id="subtabs">
                <li><a href="#mysqlguide">MySQL</a></li>
                <li><a href="#hiveguide">HIVE</a></li>
            </ul>
            <div id="mysqlguide" class="inner-tab-section">
                <h3>Tables in MySQL Database</h3>
                <table>
                    <c:forEach items="${mysqlTables}" var="mysqlTable">
                        <tr>
                            <td><a
onclick="describeMySQLTable('${mysqlTable}')"><c:out
                                value="${mysqlTable}"
/></a></td>
                        </tr>
                    </c:forEach>
                </table>
                <br />
                <div id="mysqlTableDesc"></div>
```

```
		</div>
		<div id="hiveguide" class="inner-tab-section">
			<h3>Tables in Hive Database</h3>
			<table>
				<c:forEach items="${hiveTables}" var="hiveTable">
					<tr>
						<td><a
onclick="describeHIVETable('${hiveTable}')"><c:out
value="${hiveTable}"
/></a></td>
					</tr>
				</c:forEach>
			</table>
			<br />
			<div id="hiveTableDesc"></div>
		</div>
	</div>
</body>
</html>
```

37. login.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
        pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Login Page</title>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
        <jsp:include page="header.jsp"></jsp:include>
        <%
                Cookie[] cookies = request.getCookies();
                String userName = "", password = "", rememberVal = "";
                if (cookies != null) {
                        for (Cookie cookie : cookies) {
                                if (cookie.getName().equals("cookuser")) {
                                        userName = cookie.getValue();
                                }
                                if (cookie.getName().equals("cookpass")) {
                                        password = cookie.getValue();
                                }
```

```
                        if (cookie.getName().equals("cookrem")) {
                                rememberVal = cookie.getValue();
                        }
                }
        }
    %>

    <fieldset>
        <legend>Login Credentials</legend>
        <div>
            <%=request.getAttribute("error") != null ? request
                        .getAttribute("error") : ""%>
            <form method="post"
                action="<%=request.getContextPath()%>/LoginServlet">
                <p>
                    <label class="loginLabel">Username:</label> <input
type="text"
                            class="loginInput" name="username"
autocomplete="off"
                            value="<%=userName%>" />
                </p>
                <p>
```

```
                                    <label class="loginLabel">Password:</label> <input
type="password"
                                            class="loginInput" name="password"
autocomplete="off"
                                            value="<%=password%>" />
                                </p>
                                <p>
                                    <label class="loginLabel">Remember:</label> <input
type="checkbox"
                                            class="loginInput" name="remember" value="1"
                                            <%="1".equals(rememberVal.trim()) ?
"checked=\"checked\""
                                    : ""%> />
                                </p>
                                <p>
                                    <input type="submit" class="loginButton" name="login"
value="Login" />
                                </p>
                            </form>
                        </div>
                    </fieldset>
</body>
```

```
</body>
</html>
```

38. style.css

```
fieldset {
	border: 1px solid FireBrick;
	max-width: 100%;
	margin: auto;
}
legend {
	font-weight: bold;
	font-size: .9em;
	background: white;
}
table {
	width: 100%;
	border-collapse: collapse;
}
table, th, td {
	border: 1px solid black;
}
th.thick {
```

```
        font-weight: bold;
}
.loginLabel {
        width: 150px;
        float: left;
}
.loginInput {
        transition: box-shadow 0.3s, border 0.3s;
}
#header-logo {
        max-width: 100%;
        height: auto;
}
#logout {
        font-weight: bold;
        text-decoration: underline;
        font-size: .9em;
        color: indigo;
        float: right;
}
#welcome {
        font-size: .9em;
```

```
}
#mysqlQuery, #hiveQuery, #compareQuery {
	width: 80%;
}
#mysqlexportxls, #compareexportxls, #hiveexportxls {
	width: 1.5%;
	float: right;
}
#mysqlParse, #hiveParse{
	width: 17%;
	float: right;
}
#compareParse {
	width: 15%;
	float: right;
}
#hive, #mysql, #compare, #guide{
	height: 300px;
	border: 1px solid FireBrick;
	overflow: auto;
}
#mysqlguide, #hiveguide {
```

```
        height: 180px;
        border: 1px solid FireBrick;
        overflow: auto;
}
#tabs,#subtabs {
        margin: 0;
        overflow: hidden;
        padding: 0;
        zoom: 1;
        position: relative;
        top: 2px;
        z-index: 1;
}
#tabs li, #subtabs li{
        display: block;
        list-style: none;
        margin: 0;
        margin-right: 1px;
        padding: 0;
        float: left;
}
#tabs li a ,#subtabs li a{
```

```
        display: block;
        padding: 2px 10px;
        color: black;
        border: 2px solid FireBrick;
        border-bottom: none;
        text-align: center;
        text-decoration: none;
        text-align: center;
}
.tab-section, .inner-tab-section {
        background: white;
        padding: 10px;
        border: 2px solid FireBrick;
}
#tabs li a.current, #subtabs li a.current {
        background: white;
        color: black;
        border-bottom: 2px solid white;
}
```

39. Converter.java

```
import java.io.BufferedReader;
```

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.ParseException;
import java.util.Iterator;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
public class Converter {
	public static void main(String[] args) throws org.json.simple.parser.ParseException,
IOException {
		JSONParser j=new JSONParser();
		File file = new File("/home/student/life_parsed.txt");
		if (!file.exists()) {
			try {
				file.createNewFile();
			} catch (IOException e) {
				e.printStackTrace();
			}
```

```
        }
        FileWriter fw = new FileWriter(file.getAbsoluteFile());
        BufferedWriter bw = new BufferedWriter(fw);
        try {
            BufferedReader br = new BufferedReader(new
FileReader("/home/student/life.txt"));
            String line="";
            while ((line=br.readLine()) != null ) {
                String Char = Character.toString(line.charAt(2));
                if( Char.contains("c") ) {
                    Object obj = j.parse(line);
                    JSONObject jsonObject =  (JSONObject) obj;
                                        long tweetId = (long) jsonObject.get("id");
                    String tweetCreatedAt = jsonObject.get("createdAt").toString();
                    String tweetText = (String) jsonObject.get("text");
                    if((tweetText == null ) || ((!tweetText.toLowerCase().contains("life"))
&& (!tweetText.toLowerCase().contains("people")))) {
                                                                    continue;
                    }
                    tweetText =tweetText.replaceAll("[^\\w\\s]","");
                    tweetText = tweetText.replace("\n", "").replace("\r", "");
                    Long favouriteCount = (Long) jsonObject.get("favouriteCount");
```

```
                Long retweetCount = (Long) jsonObject.get("retweetCount");
                String lang = (String) jsonObject.get("lang");
                JSONObject user= (JSONObject) jsonObject.get("user");
                Long userId = null;
                String userName = null;
                String screenName = null;
                String location = null;
                Long followersCount = null;
                Long friendsCount = null;
                String statusesCount = null;
                String timezone = null;
                if( user != null) {
                    userId = (long) user.get("id");
                    userName = (String) user.get("name");
                    screenName = (String) user.get("screenName");

userName =userName.replaceAll("[^\\w\\s]","");
                    screenName =screenName.replaceAll("[^\\w\\s]","");
                    location = (String) user.get("location");
                    followersCount = (Long) user.get("followersCount");
                    friendsCount = (Long) user.get("friendsCount");
                    statusesCount = (String) user.get("statusesCount").toString();
```

```
                timezone = (String) user.get("timeZone");
            }
            if((userName == null) || (userName.matches("^[0-
9]{11}$*&^!@#%(),.><?/"))) {
                continue;
            }
            if((screenName == null) || (screenName.matches("^[0-
9]{11}$*&^!@#%(),.><?/"))) {
                continue;
            }
            if((location==null) || (location.matches("^[0-
9]{11}$*&^!@#%(),.><?/"))) {
                continue;
            }
            if((timezone == null) || (timezone.matches("^[0-
9]{11}$*&^!@#%(),.><?/")))  {
                continue;
            }
                                                    String
fs=tweetId+"|"+tweetCreatedAt+"|"+tweetText+"|"+favouriteCount+"|"+retweetCount+"|"+lang
+"|"+userId+"|"+userN$
            bw.write(fs);
```

```
                    }
                }
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
}
```

40. TwitterFeeds.java

```
import com.google.gson.Gson;

import twitter4j.FilterQuery;
import twitter4j.StallWarning;
import java.io.*;

import twitter4j.Status;
import twitter4j.StatusDeletionNotice;
import twitter4j.StatusListener;
import twitter4j.Twitter;
import twitter4j.TwitterException;
```

```
import twitter4j.TwitterFactory;
import twitter4j.TwitterStream;
import twitter4j.TwitterStreamFactory;
import twitter4j.auth.AccessToken;
import twitter4j.conf.ConfigurationBuilder;


public class TwitterFeeds {
	public static void main(String[] args)throws NumberFormatException,
			TwitterException, IOException {
		File file = new File("/home/student/life.txt");
		// if file doesnt exists, then create it
		if (!file.exists()) {
			try {
				file.createNewFile();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
		FileWriter fw = new FileWriter(file.getAbsoluteFile());
		final BufferedWriter bw = new BufferedWriter(fw);
		ConfigurationBuilder config = new ConfigurationBuilder();
		String key = "hl2wduYX7LtGYPRyVAigeUe4o";
```

```
        String secret =
"gEwjF7J7qp8VzEfd3kEQqAWGS3wKOn2bUZwqugFVESqVq0Hjlx";
        String token = "343902303-L2rB8pf14THHA2PEzeXYa9Witq3EXJ8BTofg3TuE";
        String tokensecret = "nvb0SvwMCJoOphmqY5wn8NplW2FxoEM6iVtfqqiW40ar9";
        config.setDebugEnabled(true);
        config.setOAuthConsumerKey(key);
                        config.setOAuthConsumerSecret(secret);
        config.setOAuthAccessToken(token);
        config.setOAuthAccessTokenSecret(tokensecret);
        TwitterStream tStream = new TwitterStreamFactory(config.build())
                .getInstance();
        // Instantiate a re-usable and thread-safe factory
        TwitterFactory twitterFactory = new TwitterFactory();
        // Instantiate a new Twitter instance
        Twitter twitterSM = twitterFactory.getInstance();
        // setup OAuth Consumer Credentials
        twitterSM.setOAuthConsumer(key, secret);
        // setup OAuth Access Token
        twitterSM.setOAuthAccessToken(new AccessToken(token, tokensecret));
        StatusListener lstnr = new StatusListener() {
            @Override
            public void onException(Exception arg0) {
```

```
}
@Override
public void onDeletionNotice(StatusDeletionNotice arg0) {
}
                                    @Override
public void onScrubGeo(long arg0, long arg1) {
}
@Override
public void onStatus(Status tweetStatus) {
      String tweet = "";
      try {
            Object model = tweetStatus;
            Gson newObj = new Gson();
            tweet = newObj.toJson(model);
            tweet = tweet + "\n";
                  bw.write(tweet);
      } catch (IOException e) {
            e.printStackTrace();
      }
}
@Override
public void onTrackLimitationNotice(int arg0) {
```

```
            }
              @Override
              public void onStallWarning(StallWarning arg0) {
              }
        };
        FilterQuery filterq = new FilterQuery();
        String twittertrend[] = { "Life","People"};
        filterq.track(twittertrend);
        tStream.addListener(lstnr);
        tStream.filter(filterq);
    }
}
```